Arquitectura en Capas (continuación).

Temas de la clase: Cuestiones específicas de diseño de la capa de negocios. La capa de datos. Consideraciones generales de diseño de la capa de datos. Cuestiones específicas de diseño de la capa de datos. XML. Seguridad. Deployment (despliegue).

Cuestiones específicas de diseño de la capa de negocios (capa lógica)

Hay varias cuestiones comunes que debemos considerar mientras desarrollamos nuestro diseño. Estas cuestiones pueden ser categorizadas en áreas específicas del mismo. Aquí detallamos algunos comentarios para las áreas en las que más frecuentemente se cometen errores:

Autenticación

Diseñar una estrategia de autenticación para la capa de negocios es importante para la seguridad y confiabilidad de la aplicación. Si no lo hacemos dejamos a la aplicación vulnerable a ataques o intrusiones. Consideremos los siguientes puntos:

- Evitar la autenticación en la capa de negocios si será utilizada solo por una capa de presentación o por una capa de servicios en la misma ubicación dentro de límites de confianza. La identidad del que quiere acceder debe llegar a la capa de negocios solo si se debe autenticar o autorizar de distintas formas dependiendo de la identificación del consumidor.

- Si la capa de negocios será utilizada en múltiples aplicaciones, utilizando distintos almacenamientos de usuarios, pensemos implementar un único mecanismo de entrada (sign on)

- Si las capas de presentación y de negocios están desplegadas en la misma máquina y se debe acceder a recursos basándose en los permisos del que llama a la aplicación, podríamos usar “sustitución de identidad”, es decir, no identificar. Si el mismo caso se da con las capas ubicadas en diferentes ubicaciones consideremos el uso de “delegación” que significa “actuar en representación de…” (usar un “proxy”).

Autorización

Al igual que cuando hablamos de autenticación, no tener una estrategia de autorización, también deja a la aplicación vulnerable a que se revele la información, a que se manipulen los datos y a que se cambien privilegios. Tengamos en cuenta estas ideas:

- Proteger los recursos por medio de otorgamiento de autorizaciones a los consumidores basadas en su identidad, grupo, rol o alguna otra información de contexto.

- Utilizar autorizaciones basadas en roles para decisiones de negocio.

- Usar autorizaciones basadas en recursos para auditoría del sistema.

- Cuando se necesite basar la autorización en una mezcla de información como la identidad, permisos, etc.; utilizar autorizaciones basadas en solicitudes.

- No mezclar código referente a las autorizaciones con código referente al procesamiento en los mismos componentes.

- La infraestructura de autorizaciones no debe ser una sobrecarga para la performance.

Almacenamiento en Caché (caching)

La estrategia de uso de Caché de nuestra capa de negocio es importante para la performance de la aplicación. Usemos el almacenamiento en caché para optimizar búsquedas de datos, para evitar idas y vueltas en la red y también para no duplicar procesamiento.

Como parte de esta estrategia, debemos decidir cuándo y cómo cargar los datos del caché. Para evitar demorar al cliente, deberíamos cargarlo asincrónicamente o por medio de un proceso batch.

Es apropiado almacenar datos estáticos que serán re utilizados regularmente dentro de la capa de negocios, pero se debe evitar almacenar datos volátiles.

Se deberían almacenar los datos que no pueden ser recuperados rápida y eficientemente desde la base de datos, pero no se deben guardar grandes volúmenes de datos ya que pueden hacer mucho más lento el procesamiento. Es mejor almacenar en caché lo mínimo necesario.

Los datos almacenados en caché deben estar en un formato “listo para usar” dentro de la capa de negocios (que puedan utilizarse sin ningún tratamiento previo, como por ej. “parsearlo”).

Acoplamiento y cohesión

Cuando diseñamos los componentes para nuestra capa de negocios, deberíamos procurar la alta cohesión de los mismos (que solo se ocupen de temas relacionados entre sí) y el mínimo acoplamiento entre capas. Esto mejora la escalabilidad de la aplicación.

Para lograr esto consideremos lo siguiente:

- Evitar las dependencias circulares. La capa de negocios solo debería conocer acerca de la capa que está por debajo de ella (la de datos) y no acerca de otra que esté por encima.

- Usar la abstracción para implementar una interfaz sin acoplamiento entre capas.

- Dentro de la capa de negocio, el acoplamiento debe ser fuerte, a menos que algún comportamiento requiera lo contrario.

- Diseñar para alta cohesión. Los componentes deberían contener solo funcionalidades específicas relativas a si mismos. Siempre evite mezclar lógica de acceso a datos con lógica de negocios en la capa de negocios.

- Utilizar interfases basadas en mensajes para exponer los componentes de negocios y así reducir el acoplamiento y permitir que estén ubicados en distintos lugares físicos.

Manejo de excepciones

Una efectiva gestión de excepciones para la capa de negocios es importante para la seguridad y la confiabilidad de la aplicación.

Si esto no se tiene en cuenta, la aplicación puede quedar vulnerable a ataques tipo “denial of service” (negación de servicio) y también puede revelar información crítica.

Capturar y manejar las excepciones es una operación costosa, por lo tanto es importante que nuestro diseño de manejo de excepciones tenga en cuenta el impacto en la performance.

Tener en cuenta estas consideraciones:

- Solo se deben capturar las excepciones internas que se puedan manejar. Por ejemplo, capturar excepciones en conversiones de datos que podrían ocurrir cuando se trata de convertir valores nulos. Las excepciones no deben usarse para controlar la lógica de negocio o el flujo de la aplicación.

- La propagación de las excepciones debe ser la apropiada. Por ejemplo, permitir a las excepciones aparecer en capas lindantes, en las cuales pueden ser registradas (logging)

y transformadas como sea necesario antes de pasarlas a la siguiente capa. Se debería usar un identificador de contexto para que las excepciones puedan ser asociadas a través de las capas cuando se esté analizando la causa de errores o fallas.

- Asegurarse de que se capturan excepciones que no serán capturadas en otros lados (como ser un manejador general de errores de la aplicación).

- Diseñar una estrategia de registro (logging) y notificación para los errores críticos y excepciones. La misma debe dejar registrada información suficiente acerca de las excepciones.

Registro (logging) y auditoría

Estos temas también son importantes para la seguridad y confiabilidad de la aplicación.

Los archivos de Log, también pueden ser requeridos para probar acciones incorrectas en procedimientos legales. La auditoría se considera más autorizada (más válida) si la información del Log es generada en el momento preciso de acceso a un recurso, y por la misma rutina que accede al recurso.

La registración y la auditoría para la capa de negocios deben estar centralizadas. Se pueden implementar soluciones de terceras partes para manejarlas.

En los archivos de log no se debe guardar información sensible del negocio.

Debemos asegurarnos de que una falla en el proceso de registración no afecte el normal funcionamiento de la capa de negocios.

Validación

Una buena validación es necesaria para no dejar a la aplicación abierta a sufrir inconsistencias de datos y violaciones de las reglas del negocio, además de una mala experiencia para el usuario. También, la carencia de una estrategia de validación puede hacer que la aplicación quede insegura y pudiera sufrir ataques de intrusos.

Una estrategia de validación debería tener en cuenta lo siguiente:

- Validar todas las entradas y parámetros de los métodos dentro de la capa de negocios, aún cuando haya alguna validación en la capa de presentación.

- La validación debe estar centralizada para que sea más fácil testearla y re utilizarla.

- Restringir, rechazar y purificar lo ingresado por el usuario. En otras palabras, asumir que todo lo que ingresa el usuario es malicioso. El dato ingresado se debe validar respecto de su longitud, rango de valor, formato y tipo.

Consideraciones de despliegue o distribución (deployment)

Para el despliegue de la capa de negocios, se deben tener en cuenta los problemas de seguridad y de performance.

- La capa de negocios debería estar ubicada físicamente con la capa de presentación para maximizar la performance de la aplicación, a menos que deban estar separadas por cuestiones de escalabilidad.

- Si se debe soportar una capa de negocios remota, sería mejor usar el protocolo TCP para mejorar la performance.

- Para proteger llamadas desde los componentes de la capa de negocios a servicios web remotos (web services) utilizar encriptación SSL (secure sockets layer).

La capa de datos

La capa de datos puede incluir lo siguiente:

- **Componentes de acceso a datos**. Estos componentes abstraen la lógica requerida para acceder a los almacenamientos de datos. Centralizan funcionalidades comunes de acceso a datos para hacer que la aplicación sea más fácil de configurar y de mantener.

- **Agentes de servicio**. Cuando un componente de negocio debe acceder a datos provistos por un servicio externo, deberíamos necesitar implementar código para manejar la comunicación con ese servicio en particular. Los agentes de servicio implementan componentes de acceso a datos que aíslan los variados requerimientos necesarios para las llamadas a servicios desde nuestra aplicación.

Consideraciones generales de diseño de la capa de datos

La capa de datos debe cumplir con los requerimientos de la aplicación, ejecutar de forma eficiente y segura, y ser fácil de mantener y de ampliar a medida que vayan cambiando los requerimientos del negocio.

Cuando la diseñamos deberíamos considerar lo siguiente:

- Elegir una apropiada tecnología de acceso a datos. La elección de la tecnología depende del tipo de datos que se deben manejar y cómo se manipularán los mismos dentro de la aplicación.

- Usar abstracción para implementar una interfaz independiente para acceder a la capa de datos. Esto puede hacerse definiendo componentes de interfaz tales como "puertas de entrada", con entradas y salidas que traduzcan las solicitudes a un formato que pueda ser entendido por los componentes dentro de la capa.

- Encapsular las funcionalidades de acceso a datos dentro de la capa. Esta capa debería ocultar los detalles de acceso a las fuentes de datos. Debe ser la responsable de administrar las conexiones, de generar las consultas, etc.

- Los consumidores de la capa de acceso a datos interactúan a través de interfases abstractas usando objetos de cliente, como datasets y XML, y no deberían conocer nada acerca de los detalles internos de la capa de acceso a datos.

- Decidir cómo se manejarán las conexiones. Como regla, la capa de acceso a datos debería crear y manejar todas las conexiones con todas las fuentes de datos requeridas por la aplicación. Se debe decidir un método apropiado para almacenar y proteger la información de las conexiones, quizás encriptando secciones del archivo de configuración o guardar esa información en el servidor para cumplir con los requerimientos de seguridad de la empresa.

- Determinar cómo se manejarán las excepciones relativas a datos. La capa debería capturar y manejar (por lo menos inicialmente) todas las excepciones asociadas con las fuentes de datos y con las operaciones de creación, lectura, actualización y eliminación. Las excepciones referentes a los datos mismos, y al acceso a fuentes de datos y los errores de tiempo excedido (time out), deben ser administrados en esta capa y luego ser transferidos a otras capas solo si las fallas afectan a las aplicaciones.

- Considerar los riesgos de seguridad. La capa debe protegerse contra posibles ataques que traten de robar o de corromper los datos. Se restringe el otorgamiento de los permisos a solo aquellos que necesiten ejecutar las operaciones requeridas por la aplicación. No se deben usar concatenaciones de strings para construir consultas dinámicas a partir de los datos ingresados por el usuario.

- Reducir las idas y vueltas. Considerar el uso de comandos batch para una única operación en la base de datos.

- Pensar en la performance y en la escalabilidad. Estos deberían ser objetivos a cumplir por la capa de datos.

Cuestiones específicas del diseño

Mientras diseñamos la capa de datos debemos tener en cuenta algunas cuestiones comunes. Estas cuestiones pueden ser categorizadas en áreas específicas:

Tareas Batch

El procesamiento batch de comandos de la base de datos, puede mejorar la performance de la capa de datos. Cada pedido de ejecución a la base de datos incurre en una sobrecarga. El procesamiento batch de consultas similares puede mejorar la performance porque la base de datos almacena en caché y reutiliza un plan de ejecución de una consulta para otra consulta similar.

Tener en cuenta:

- El uso de comandos batch reduce las idas y vueltas hacia la base de datos y minimiza el tráfico de red. Sin embargo, solo se aconseja ejecutar en modo batch las consultas similares.

- Es mejor usar comandos batch para cargar o copiar múltiples conjuntos de datos.

- No se deben ejecutar transacciones que lockearán recursos de la base de datos en largos procesos batch.

Grandes objetos binarios

Cuando los datos son almacenados y recuperados como una sola unidad (single stream), podemos pensarlos como que son un gran objeto binario o BLOB (binary large object). Un BLOB puede tener estructura interna, pero esta estructura no se manifiesta a la base de datos que lo guarda (no ve esa estructura), o a la capa de datos que lo lee y lo graba. Las bases de datos pueden guardar los datos del BLOB o punteros que apunten a ellos. Si el BLOB no está guardado directamente en la base de datos, estará en el sistema de archivos. Los BLOB se usan generalmente para guardar imágenes.

- Considerar si se necesita guardar el BLOB en la base de datos. Las bases de datos modernas tienen muy buen manejo de datos BLOB, proporcionando apropiados tipos de datos para el campo y también versionamiento. Sin embargo se debe evaluar si es más práctico guardar los datos en el disco y solo guardar un vínculo a los datos en la base de datos.

- El uso de BLOBs simplifica la sincronización de grandes objetos binarios entre servidores.

- Si va a ser necesario buscar datos BLOB, será mejor crear y cargar otros campos de la base de datos que puedan usarse para búsquedas, en vez de "parsear" los datos del BLOB.

- Cuando se recuperan datos de BLOB, los mismos deben tomar la forma apropiada ("cast" de los datos, se refiere a conversión de tipos de datos) para que puedan ser manipulados dentro de la capa de negocios o de presentación.

Conexiones

Todas las conexiones con las fuentes de datos deberían ser administradas por la capa de datos. La creación y gestión de las conexiones utilizan recursos valiosos tanto en la capa de datos como en la fuente de datos. Para maximizar performance y seguridad consideremos lo siguiente:

- En general, se deben abrir las conexiones lo más tarde posible y cerrarlas lo más temprano posible. Nunca debemos mantener conexiones abiertas durante períodos excesivos.
- Ejecutar transacciones a través de una sola conexión siempre que se pueda.

Formato de datos

La correcta elección del formato de datos proporciona interoperabilidad con otras aplicaciones. El formato de datos y la serialización son importantes porque permiten el almacenamiento y recuperación del estado de una aplicación por parte de la capa de negocios (serializar es convertir el estado de una estructura de datos o de un objeto a un formato en el que pueda ser guardado, y luego puede ser "resucitado" en otro entorno)

Es mejor utilizar XML para interoperabilidad con otros sistemas y plataformas, o cuando se trabaja con estructuras de datos que pueden ir cambiando con el tiempo.

Administración de excepciones

El manejo de excepciones debe ser centralizado, de modo que las excepciones sean capturadas y enviadas consistentemente a nuestra capa de datos. Si es posible, es mejor centralizar el manejo de excepciones en componentes que implementan concernimientos transversales (mencionados en la clase anterior). Se debe prestar especial atención a las excepciones que se propagan a otras capas.

- Identificar las excepciones que deberían ser capturadas en la capa de datos. Por ejemplo abrazos mortales y problemas de conexión pueden ser resueltos dentro de esta capa.

- Como ya dijimos cuando hablamos del diseño de la capa de negocios, se debe permitir que las excepciones se propaguen a capas lindantes donde puedan ser registradas.

- Implementar procesos de reintento para las operaciones cuando ocurren errores de las fuentes de datos o errores de tiempo excedido (time out), siempre y cuando sea seguro hacerlo.

- Como también dijimos antes, asegurarnos de que las excepciones que se capturen no sean también capturadas por un manejador de errores general de la aplicación.

Consultas

Las consultas son las operaciones primarias de manipulación de datos de esta capa. Son el mecanismo que traduce las solicitudes de la aplicación en acciones de creación, lectura, actualización y eliminación en la base de datos (CRUD actions). Como las consultas son tan importantes, deberían ser optimizadas para maximizar la performance y el rendimiento. Para ello tengamos en cuenta lo siguiente:

- Usar consultas SQL parametrizadas para prevenir problemas de seguridad y reducir la chance de que ocurran ataques tipo “inyección de sql” (inserción de código sql invasor dentro del código sql programado). Muchas veces usamos concatenaciones de strings para construir consultas dinámicas a partir de los datos ingresados por el usuario. Si hacemos eso en vez de ejecutar consultas parametrizadas, las bases de datos no son capaces de reutilizar el plan de ejecución de consultas similares y entonces al costo de la ejecución de la consulta hay que sumarle el de buscar el plan de ejecución adecuado. Cuando usamos consultas parametrizadas, en vez de concatenar el valor de variables como cadenas, escribimos las sentencias utilizando parámetros y luego ejecutamos las consultas asignando previamente valores a los parámetros. De ese modo la base de datos verá siempre la misma consulta y podrá reutilizar el plan de ejecución.

- Cuando se construyen consultas SQL dinámicas, evitar mezclar lógica de procesamiento de negocio con lógica utilizada para generar la instrucción SQL, ya que eso lleva a que el código sea muy difícil de mantener.

Procedimientos almacenados (stored procedures)

En el pasado, los procedimientos almacenados representaban una mejora de performance sobre las instrucciones SQL dinámicas. Sin embargo, con los modernos manejadores de bases de datos, la performance de los procedimientos almacenados y de las instrucciones SQL dinámicas son similares.

En términos de seguridad y de performance, una primera regla con los “stored procedures” es usar parámetros y evitar SQL dinámico dentro de los mismos. Debemos tener en cuenta que una ventaja de los procedimientos almacenados es que se compilan una sola vez (solo la

primera vez que se ejecutan), pero si los usamos con sql dinámico, esto no se cumple y deben recompilarse para cada ejecución, lo que reduce la performance.

Los parámetros son uno de los factores que llevan al uso de los planes de ejecución de consultas almacenados en caché en vez de reconstruir el plan de consulta desde cero. Cuando el tipo y el número de parámetros cambian, se generan nuevos planes de ejecución de consultas, lo que reduce la performance.

Consideraciones sobre el uso de procedimientos almacenados:

- Usar parámetros escritos como valores de entrada al procedimiento y parámetros de salida para retornar valores únicos. Usar parámetros XML y no cambiar los datos a formato para mostrar ("display format") en el procedimiento almacenado. En vez de eso, retornar el tipo apropiado y ejecutar el cambio de formato en la capa de presentación.

- Si se va a generar SQL dinámico dentro del procedimiento usar parámetros.

- Evitar la creación de tablas temporales durante el procesamiento. Si estas deben ser utilizadas, es mejor crearlas en memoria en lugar de crearlas en el disco.

- Implementar manejo de errores y retornar errores que el código de la aplicación pueda manejar.

Procedimientos almacenados vs. SQL dinámico

La elección entre ellos se enfoca principalmente en el uso de instrucciones SQL generadas dinámicamente en código en vez de SQL implementado dentro de un stored procedure.

Para optar entre ambos debemos considerar los requerimientos de abstracción, mantenimiento y limitaciones del entorno. También, esta elección está relacionada con las preferencias o conocimientos del desarrollador.

La principal ventaja de los procedimientos almacenados es que proporcionan una capa de abstracción a la base de datos que minimiza el impacto en el código de la aplicación cuando cambia el esquema de la base de datos. La seguridad también es más fácil de implementar porque se puede restringir el acceso a todo excepto al procedimiento, y tomar ventaja de las características de seguridad soportadas por la mayoría de las bases de datos.

La principal ventaja de las instrucciones SQL dinámicas es que son consideradas más flexibles que los procedimientos almacenados y permiten un desarrollo más rápido. Incluso hay generadores de consultas dinámicas que hacen el trabajo por nosotros.

Para optar por una o por otra aproximación, tener en cuenta:

- Si la aplicación es pequeña y tiene un único cliente y pocas reglas de negocio, SQL dinámico es en general la mejor opción.

- En una aplicación grande con múltiples clientes (aplicaciones cliente), consideremos como se puede alcanzar la abstracción requerida. Hay que decidir en donde va a estar la abstracción: en la base de datos como procedimiento almacenado o en la capa de datos en la forma de patrones de acceso.

- Para operaciones intensivas de datos, los procedimientos permiten ejecutar las operaciones bien cerca de los datos, lo que mejora la performance.

- Para minimizar los cambios en el código de la aplicación cuando cambie el esquema de la base de datos, se debería usar procedimientos. Los cambios en un procedimiento almacenado pueden afectar el código de la aplicación, pero esos cambios pueden afectar solo a componentes específicos que acceden al procedimiento almacenado.

- Si se utilizan consultas SQL dinámicas, se debe entender el impacto que los cambios en el esquema de la base de datos tendrán sobre la aplicación. Por lo tanto, se debería implementar la capa de acceso a datos en una forma en que no se acoplen componentes de negocio con la ejecución de consultas a la base de datos.

- Las consultas SQL dinámicas son más fáciles de depurar (debugging)

Transacciones

Una transacción es un intercambio de información secuencial y acciones asociadas que son tratadas como una unidad atómica, para satisfacer una solicitud y asegurar la integridad de la base de datos. Se considera completa solo si toda la información y las acciones están completas y los cambios en la base de datos están hechos. Las transacciones soportan “deshacer” las acciones luego de un error (rollback), lo cual ayuda a preservar la integridad de los datos.

Es importante usar el modelo de concurrencia apropiado y determinar cómo se manejarán las transacciones. Se puede optar entre un modelo optimista y uno pesimista para la administración de la concurrencia.

Con el manejo optimista, los datos no se lockean y la actualización necesita chequear que los datos no han cambiado desde que fueron leídos por última vez.

Con el manejo pesimista, los datos son lockeados y no pueden ser actualizados por otra operación hasta que el lockeo sea liberado.

Entonces debemos tener en cuenta:

- Habilitar transacciones solo cuando se necesite. Consultas simples pueden no necesitar una transacción explícita, pero debemos estar seguros de conocer el comportamiento del "commit" por defecto de nuestra base de datos (por ejemplo Microsoft SQL SERVER considera cada instrucción SQL como una transacción individual, en modo de transacción auto-commit)

- Mantener las transacciones lo más breves posibles para minimizar la cantidad de tiempo que los lockeos son mantenidos. Tratar de evitar el uso de lockeos para transacciones de larga duración. No usar lockeos exclusivos ya que pueden causar abrazos mortales (deadlocks).

- Si bien las transacciones implícitas son más lentas que las manuales (o explícitas), son más fáciles de desarrollar y de mantener. Si se usan transacciones manuales, es mejor implementarlas dentro de un procedimiento almacenado.

Validación

La estrategia de validación es crítica para la seguridad de la aplicación. Debemos determinar las reglas de validación para los datos recibidos de otras capas y de componentes de terceras partes, así como también de la base de datos.

- Validar todos los datos recibidos por la capa de datos desde todos los consumidores. Asegurarse de manejar correctamente los valores nulos y de filtrar caracteres inválidos.

- Retornar mensajes de error claros e informativos si la validación falla.

XML

El lenguaje de marcado XML es muy útil para la interoperabilidad y para el mantenimiento de la estructura de datos fuera de la base de datos. Por razones de performance, se debe ser cuidadoso al usar XML para volúmenes de datos muy grandes.

Si se debe manejar grandes volúmenes con XML, usar esquemas basados en atributos, donde los valores de los datos son guardados como atributos, en vez de esquemas basados en elementos, que guardan los valores de datos como valores de elementos y son consecuentemente más grandes.

Consideraciones de diseño para el uso de XML:

- Utilizar lectores y escritores de XML para acceder a datos bajo el formato XML, especialmente para grandes volúmenes de datos.

- Usar un esquema XML para definir formatos y proporcionar validación para los datos guardados y transmitidos como XML (aunque esto afecta la performance).

Consideraciones de performance

La performance es función tanto del diseño de la capa de datos, como del diseño de la base de datos. Debemos considerarlas juntas cuando hagamos ajuste fino (tunning) del sistema para un máximo rendimiento de los datos.

Para diseñar pensando en la performance, tener en cuenta lo siguiente:

- Usar pool de conexiones (colección de conexiones abiertas a una base de datos para que puedan ser reutilizadas) y ajustar la performance basándose en los resultados obtenidos en escenarios simulados.

- Utilizar comandos batch para reducir el número de idas y venidas al servidor de base de datos.

- Consideremos usar concurrencia optimista para minimizar el costo del lockeo (las conexiones deben permanecer abiertas durante un lockeo).

Consideraciones de seguridad

La capa de datos debe proteger la base de datos contra ataques que quieran robar o corromper los datos. Pensando en la seguridad en nuestro diseño, tener en cuenta lo siguiente:

- Encriptar los strings de conexión en los archivos de configuración.

- Al guardar contraseñas, usar un algoritmo de hash en vez de una versión encriptada de la contraseña.

- Requerir a los consumidores (llamadores) que envíen información de identidad a la capa de datos para fines de auditoría.

Consideraciones de distribución (deployment)

Al desplegar la capa de datos, la meta del arquitecto de software es la buena performance y la seguridad en el entorno de producción.

Consideremos las siguientes ideas:

- Ubicar la capa de acceso a datos en la misma ubicación que la capa de negocio para mejorar la performance, a menos que esto vaya en contra de la escalabilidad.

- La capa de datos debería estar en un servidor distinto del que aloja a la base de datos. Las características físicas de un servidor de base de datos están optimizadas solo para ese rol y difícilmente esa optimización sirva también para una óptima operación de la capa de datos. La combinación de ambos en una misma ubicación reduce la performance de la aplicación.