José Mérida, Luis Padilla

18-05-2025

Bases de Datos 1

Laboratorio 2

1. ¿Qué ventajas encontraron al encapsular lógica en funciones en lugar de repetir consultas SQL?

La ventaja principal que encontramos fue tener un único punto de verdad, dónde por ejemplo la función de calcular tiempo promedio de adopción se puede utilizar en múltiples lugares. En el caso que no tuviéramos una función definida, si necesitáramos cambiar la lógica de negocio (por ejemplo, alguna tabla cambió o estamos almacenando algún valor de otra manera) necesitaríamos ir a cada uno de los lugares dónde se realizó la consulta. Lo mismo aplica para verificar si un animal se puede adoptar, al definir una función no necesitaríamos ir a cambiar los criterios de adopción a diferentes vistas o funciones. Luego, creo que es de bastante ayuda también para sistemas que van a crecer y se evita tener que escribir mucho código boilerplate. Se vuelve algo tedioso dónde cada join / where se tiene que verificar al momento de generar queries complejos como lo fueron los del proyecto 3. También en cuánto a legibilidad encontramos una ventaja, en vez de ver:

“WHERE NOT animal.adoptado AND animal.salud != ‘Crítica’”

Podemos ver

“animal_puede_ser_adoptado”

Dejando muy claro lo que se está haciendo.

2. ¿Qué criterios usaron para decidir cuándo implementar una función y cuándo una vista?

Nos basamos principalmente en tres cosas, primero las vistas no pueden retornar escalares. Al momento de querer calcular el tiempo promedio de adopción dentro del refugio, decidimos hacerlo una función para poder utilizar ese escalar directamente en diferentes lugares. Segundo, las vistas no aceptan parámetros. Debido a esto, decidimos utilizar una función para filtrar animales en adopción por especie de manera

más fácil. Por último, nos basamos en las responsabilidades de cada uno. Dónde priorizamos tener las reglas de negocio encapsuladas dentro de una función, cómo puede ser definir la elegibilidad para adopción de cada animal dentro de una función y no hard-coded dentro de una vista.

3. ¿Qué limitaciones encontraron al trabajar con procedimientos almacenados en comparación con funciones?

Los procedimientos almacenados son bastante menos flexibles que las funciones, ya que no devuelven algún valor. Algunas de las funciones que definimos las utilizamos para calcular ciertos valores, por ejemplo, y esto no se puede hacer con procedimientos almacenados. Otra desventaja que realmente no experimentamos, pero debemos de tomar en cuenta, es que se ejecutan de manera explícita y no se integran cómo consultas de SQL. Por último, existe una diferencia grande en cuánto al manejo de transacciones que puede llegar a ser una ventaja o desventaja. Los procedimientos almacenados permiten utilizar transacciones, mientras que las funciones siempre se llevan a cabo en el contexto que las invoca. Debido a esto, hay que tener bastante cuidado con la lógica que se coloca dentro de las funciones y verificar que se hagan dentro de una transacción en caso de ser funciones con múltiples pasos.

4. ¿Creen que el trigger que implementaron garantiza la integridad de los datos en todos los escenarios posibles? Justifiquen su respuesta

Si, en realidad los triggers que implementamos son bastante generales y debido a eso robustos. El sistema no es sumamente complejo y los triggers realizan acciones cómo revisar adoptabilidad de un animal o eliminar de algunas tablas luego de un delete. Realmente no veo algún escenario dónde se le puedan hacer baipás a los triggers que implementamos.

5. ¿Cómo adaptarían su solución para que escale en una base de datos con millones de registros?

Probablemente normalizaríamos más las tablas, así evitamos almacenar información redundante. Con más normalización probablemente tendríamos que rediseñar los

triggers y restricciones que diseñamos en el laboratorio. Además, podríamos indexear las tablas para búsquedas más rápidas.

6. ¿Qué escenarios podrían romper su lógica actual si no existiera el trigger?

Los dos más claros serían,

- Se adopta un animal en condición crítica o que no se encuentre en adopción, esto puede resultar en dos adopciones de un mismo animal y romper la lógica del sistema. Se previene por medio del trigger al realizar una validación antes de insertar.

- Un animal continúa siendo marcado como adoptado a pesar de no estarlo, en este caso el animal queda en “limbo” incapaz de ser visualizado por los filtros de adopción.

7. ¿Qué dificultades enfrentaron al definir funciones que devuelven conjuntos de resultados?

Realmente ninguna, esto puede ser debido a que no las utilizamos tanto cómo las que devuelven un valor único. La única como “desventaja” que podría decir, sería que en casos que se use cómo reportería son bastante inferiores a algún view al menos que se necesite pasar algún parámetro.

8. ¿Consideran que su diseño sería compatible con una arquitectura de microservicios? ¿Por qué sí o por qué no?

Hasta donde tenemos entendido, tener una base de datos con lógica y datos centralizados no es muy compatible con la arquitectura de microservicios. En este caso, la información que almacenamos está relacionada muy cercanamente y se nos dificulta pensar en cómo podría ser compatible separarla. Además, toda la lógica de negocios se encuentra centralizada en la base de datos y normalmente involucra diferentes tablas (dónde, por ejemplo, tenemos que movernos de adopciones a animales para marcarlos cómo adoptados).

9. ¿Cómo reutilizarían las vistas que definieron en reportes o en otros sistemas?

Nuestras vistas principalmente se basan en realizar pequeños reportes para la visualización de datos, especialmente por tener “fresco” el proyecto anterior que era una aplicación de reportería. Pensamos en algunos queries que podrían correrse frecuentemente, cómo lo podría ser ver animales en adopción o estadísticas agrupadas por especie. Luego, consideramos importante también poder tener vistas individuales de cada animal para poder tener su historial. Y por último, implementamos una vista que calcula los pagos hacia los veterinarios y cuánto han costado sus procedimientos. Consideramos que estas vistas pueden ser utilizadas dentro de un módulo de reportes y business intelligence dentro de una aplicación.

10. ¿Qué aprendieron sobre la separación entre la lógica de negocio y la lógica de persistencia al hacer este laboratorio?

El manejo de la lógica de almacenamiento y lógica de negocio debe depender bastante de la arquitectura que se esté siguiendo para la aplicación. En este laboratorio, por ejemplo, definimos algunas funciones incluyen reglas de negocio con algunos checks y validaciones con la finalidad de mantener la integridad de los datos. Esto puede ser tanto un positivo cómo un negativo; por una parte, la lógica de negocios está definida dentro de funciones y se puede implementar dentro de diferentes partes dentro de la misma base de datos, sin embargo, al momento de modificar las reglas de negocio se deben modificar tanto en la base de datos cómo en la capa de aplicación. Consideramos que un sistema monolítico podría implementar algunas de las validaciones que implementamos en el laboratorio, sin embargo, esto lo hace poco apto para una arquitectura de microservicios. Por esto es importante tener claro las consideraciones fuera del sistema de base de datos en la etapa de diseño.