Who Needs Stored Procedures, Anyways?

https://blog.codinghorror.com/who-needs-stored-procedures-anyways/

El siguiente articulo trata sobre el uso o no de los store procedure a la hora de realizar operaciones en una base de datos, para ellos hace un paralelo a las desventajas y los beneficios que estos traen

Primeramente habla sobre las desventajas que los store procedure traen, hablando como primero sobre que estos ya son un lenguaje arcaico y están llenos de opciones de diseño alocadas e incoherentes que siempre resultan de la tortuosa evolución de diez años de compatibilidad con versiones anteriores, que este tipo de lenguaje ve vuelve engorroso a la hora de tratar de corregir un error ya que los errores que emite Oracle sobre estos procedimientos no son detallados por lo cual se debe de realizar una búsqueda detallada sobre cada línea de código para poder hallar el error; otra de sus falencias es que estos no permiten pasar objetos y por otro lado los store procedure no permiten ver la lógica de negocio no permitiendo ver lo que se esta haciendo en un proceso.

Pero no todo es malo, ya que este tipo de lenguaje también tiene sus beneficios, los cuales van muy enfocados a lo que es el rendimiento de la base de datos y las consultas, ya que estos quedan almacenados en la memoria chache, permitiendo nuevamente su acceso de una manera más rápida a la hora de ejecutar.

Estos también proporcionan beneficios de seguridad, ya que solo se otorgan permisos desde la Base de datos a ciertos usuarios ara ejecutarlos, mas no permiso para acceder a las tablas subyacentes.

Al mismo tiempo también permiten reducir el trafico en una red, ya que se pueden ejecutar por lotes en lugar de enviar múltiples solicitudes.

Sin embargo, hasta el momento de la práctica esto no se logra concebir. Los beneficios son marginales pero el dolor es sustancial.

En la actualidad la arquitectura de los store procedure tienen muchas desventajas y pocos beneficios. Debe de considerarse como un lenguaje de ensamblaje de la base de datos, solo para usarse en un momento critico en las que se necesite un mayor rendimiento.

Maybe Normalizing Isn't Normal

https://blog.codinghorror.com/maybe-normalizing-isnt-normal/

este articulo trata sobre la normalización en las bases de datos, de lo cual habla que la normalización, va dirigida a limitar la duplicidad de datos, por lo que, por lo que casi no hay riesgo de inconsistencia en los datos, pero a la hora de realizar una consulta se requieren muchas uniones (joins) para poder recuperar información.

La normalización puede generalmente causar lentitud, lo cual no ayuda en nada al rendimiento de un sistema, y es por ello que se plantea el termino Desnormalizar.

Normalizar o desnormalizar son términos iguales en el momento en que posean pocos datos en una base de datos, pero cuando se poseen millones de estos datos, es ahí en donde entramos a comprender estos términos, a lo cual se puede ver afectado el rendimiento de la misma base de datos a la hora de tratar de acceder a la misma y querer recuperar cierta información necesaria, para lo cual la desnormalización no es lo más adecuado.

Cuando se trata de la base de datos debemos medir el rendimiento de esta, tratando de errar tanto en el diseño simple (desnormalizado), como en el diseño sensato (normalizado), se debe de elegir el diseño que mas se ajuste a las necesidades, el mas sencillo y familiar. Se puede desnormalizar en donde tiene sentido hacerlo y mantener normalizado lo que es necesario tener así. Escoger el diseño depende de las necesidades que la base de datos arroje.

Para diseñar una base de datos nonos debemos basar en conocimientos entregados por otros a la hora de decidir que tanto de debe normalizar o desnormalizar, puesto que es la misma base de datos la que nos proporcionara es información:

1. La normalización es lo mas viable para el sistema.
2. La normalización proporciona un mejor rendimiento.
3. La normalización evita una cantidad engorrosa de duplicidad de datos.
4. La normalización permite escribir consultas o código más simple.

Para todo ellos debo medir el rendimiento y decidir hasta que punto debo normalizar o desnormalizar, teniendo en cuenta que ahora los discos y memorias son mas baratos y de mayor capacidad.

10 Reasons To Consider A Multi Model Database

http://highscalability.com/blog/2015/3/4/10-reasons-to-consider-a-multi-model-database.html

Este tipo de base de datos elimina la fragmentación y proporciona un Back-end coherente y bien entendido, que admite muchos productos y aplicaciones diferentes.

Para ellos vemos 10 beneficios de las bases de datos de modelos múltiples.

1. **Consolidación:** Admite diferentes tipos de datos y casos de uso para consolidarlos en una sola plataforma, ganando flexibilidad en el lenguaje de consultas y el modelo de datos, y al mismo tiempo se beneficia de una tecnología de almacenamiento común.

2. **Escalado o rendimiento:** Este modelo permite el escalonamiento independiente, escalando diferentes componentes por separado, dentro de una misma arquitectura, a medida que cambian las necesidades, en vez de hacerlo verticalmente en una sola máquina.

3. **Complejidad operacional:** Varias bases de datos aumentan la complejidad operacional y de desarrollo de un sistema o aplicación. El objetivo de la persistencia poliglota es utilizar el mejor componente para el trabajo, terminar al fin con varias bases de datos, pero cada una de ella con sus propios requisitos operativos y de almacenamiento.

4. **Flexibilidad:** En vez de tener muchos datos en una sola modelo de base de datos, el enfoque multimodelo, nos permite mapear varios modelos de un único motor de almacenamiento que puede admitir diferentes casos de uso y aplicaciones.

5. **Confiabilidad:** Al ejecutar varias bases de datos podemos tener muchos problemas de fallos para el sistema y las aplicaciones de mayor tamaño, y para algunos sistemas l recuperación de fallas puede conllevar mucho tiempo, costos tremendos y experiencias engorrosas de las aplicaciones.

6. **Consistencia de los datos:** N hay una manera de mantener la coherencia de los datos entre diferentes modelos, para lo cual se requieren transacciones ACID para los diferentes tipos de datos, para que así funcionen en los diferentes modelos, pero con un único sistema de back-end que admita múltiples modelos, puede lograr este objetivo.

7. **Tolerancia a Fallas:** Un sistema con muchos componentes que sea tolerante a fallas no es fácil de obtener, y una integración de múltiples sistemas tolerantes a fallas de todo el

sistema es muy costoso, pero desafortunadamente esto se hace necesario para que un sistema funcione correctamente.

8. **Costos:** A mayor uso de los sistemas de bases de datos aumentan los costos según las necesidades. En conjunto los sistemas focalizados para resolver problemas comerciales específicos suman costos muy rápidamente, incluyendo actualizaciones, parches, correcciones de errores, otras modificaciones y pruebas necesarias de los componentes.

9. **Transacciones:** Los sistemas de las bases de datos relacionales, generalmente se implementan en una sola máquina, ofreciendo garantías transaccionales para las operaciones de la base de datos, lo cual facilita comprender con certeza el estado actual de la base de datos, lo cual las bases de datos NoSQL no ofrecen estas garantías.

10. **Mejores aplicaciones:** Ejecutar aplicaciones en una base de datos de modelos múltiples obtiene beneficios de escalabilidad, tolerancia a fallas y alto rendimiento integrado al producto.

Es hacia esto a donde se dirige el mercado de bases de datos, transacciones con ACID, API de modelos múltiples y motores de almacenamiento compartido y potentes.