**WHO NEEDS STORED PROCEDURES, ANYWAYS**

**(Autor: Jeff Atwood)**

**(tomado: https://blog.codinghorror.com/maybe-normalizing-isnt-normal/)**

El autor indica con intencionado sarcasmo, que la entrada diaria de WTF en procedimientos almacenados debe tomarse al pie de la letra, donde indica que siempre se ha sabido que la línea de SQL es una mala práctica y que debe usarse procedimientos almacenados cuando sea posible.

Algunas desventajas al utilizar los procedimientos almacenados son los siguientes:

* Los procedimientos almacenados están escritos en grandes "idiomas" de bases de datos fuertes, como PL / SQL (Oracle) o T - SQL (Microsoft). Estos lenguajes son antiguos y están llenos de diseño incoherente que siempre resultan de tortuosa evolución de diez años de compatibilidad con versiones anteriores. por argumento JavaScript es lenguaje que va un paso adelante que los mencionados lenguajes.
* Los procedimientos almacenados no proporcionan comentarios cuando las cosas van mal
* Los procedimientos almacenados no pueden pasar objetos. Entonces, si no tiene cuidado, puede terminar con un trillón de parámetros. Si tiene que llenar una fila de la tabla con más de 20 campos usando un proceso, digamos hola a más de 20 parámetros. Lo peor de todo es que si paso un parámetro incorrecto, ya sea demasiados, no suficientes o malos tipos de datos, aparece un error genérico de "llamada incorrecta". Oracle no puede decirme qué parámetros son erróneos. Así que tengo que estudiar más de 20 parámetros, a mano, para descubrir cuál es el culpable.

Entonces, ¿por qué utilizar los procedimientos almacenados? La sabiduría popular dice que lo hacemos porque:

* Los procedimientos almacenados generalmente dan como resultado un mejor rendimiento porque la base de datos puede optimizar el plan de acceso a datos utilizado por el procedimiento y almacenarlo en caché para su reutilización posterior.
* Los procedimientos almacenados se pueden proteger individualmente dentro de la base de datos. A un cliente se le pueden otorgar permisos para ejecutar un procedimiento almacenado sin tener ningún permiso en las tablas subyacentes.
* Los procedimientos almacenados presentan un mantenimiento más fácil porque es más sencillo modificar un procedimiento almacenado que cambiar una declaración SQL codificada dentro de un componente desplegado.
* Los procedimientos almacenados pueden reducir el tráfico de la red, ya que las sentencias SQL se pueden ejecutar en lotes en lugar de enviar múltiples solicitudes desde el cliente.

El autor concluye, que hay muchas maneras de diseñar una capa de acceso a datos sólida y de alto rendimiento sin recurrir a procedimientos almacenados; obtendrá muchos beneficios si se queda con SQL parametrizado y un único entorno de desarrollo coherente.

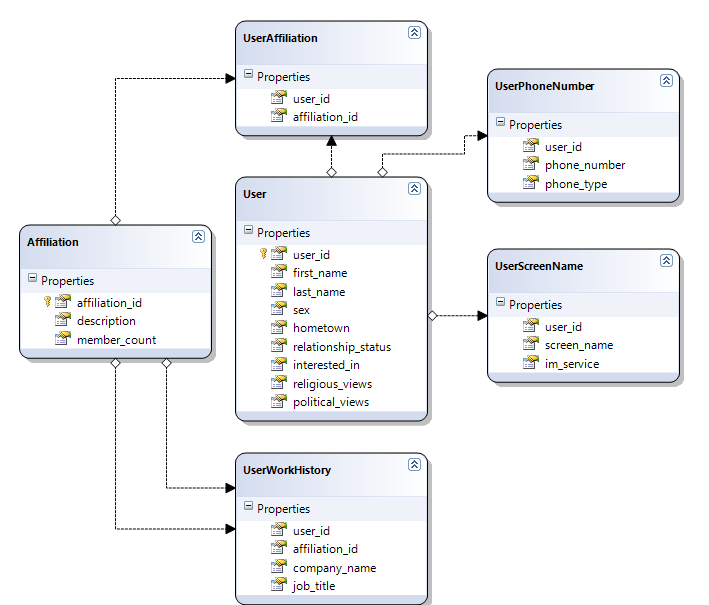
**MAYBE NORMALIZING ISN'T NORMAL**

#### (Autor: Jeff Atwood)

#### (Tomado: https://blog.codinghorror.com/who-needs-stored-procedures-anyways/)

El autor indica que uno de los elementos con los que ahora se lucha en Stack Overflow es cómo mantener niveles de rendimiento casi instantáneos en una base de datos relacional a medida que aumenta la cantidad de datos.

El autor hace referencia a Dare Obasanjo quien tuvo una excelente publicación *When Not to Normalize your SQL Database* en la que proporciona útilmente un esquema de muestra de base de datos para un sitio genérico de redes sociales. La cual se muestra en la siguiente figura:



La normalización sin duda cumple en términos de limitar la duplicación. Pero este diseño también requiere de las grandes seis combinaciones para recuperar la información de un solo usuario.

Como señala Dare, la solución obvia es desnormalizar: congestionar una gran cantidad de datos en una sola tabla de Usuarios.

Las consultas ahora son increíblemente simples (select \* from users), y probablemente también son increíblemente rápidas.

El autor afirma que ambas soluciones tienen sus pros y sus contras, y hace la gran pregunta: ¿qué es mejor, una base de datos normalizada o una base de datos desnormalizada?

Jeff Atwood con su humor negro responde: *¡Pregunta capciosa! ¡La respuesta es que no importa!* Hasta que tengas millones y millones de filas de datos, eso es. Incluso una computadora sencilla para los estándares actuales, digamos de doble núcleo con 4 gigabytes de memoria, le brindará un rendimiento casi idéntico en cualquier caso para cualquier cosa que no sea la base de datos más grande.

En estos casos, existen historias fascinantes sobre la guerra de base de datos de compañía los cuales han sido compilados por Tim O'Reilly y el blog blog High Scalability.

* [Second Life](http://radar.oreilly.com/2006/04/web-20-and-databases-part-1-se.html)
* [Blogline and Memeorandum](http://radar.oreilly.com/2006/04/database-war-stories-2-bloglin.html)
* [Flickr](http://radar.oreilly.com/2006/04/database-war-stories-3-flickr.html)
* [NASA World Wind](http://radar.oreilly.com/2006/04/database-war-stories-4-nasa-wo.html)
* [Craigslist](http://radar.oreilly.com/2006/04/database-war-stories-5-craigsl.html)
* [O'Reilly Research](http://radar.oreilly.com/2006/05/database-war-stories-6-oreilly.html)
* [Google File System and BigTable](http://radar.oreilly.com/2006/05/database-war-stories-7-google.html)
* [Findory and Amazon](http://radar.oreilly.com/2006/05/database-war-stories-8-findory.html)
* [MySQL](http://radar.oreilly.com/2006/05/database-war-stories-9-finis-b.html)
* [YouTube](http://highscalability.com/youtube-architecture)
* [PlentyOfFish](http://highscalability.com/plentyoffish-architecture)
* [Google](http://highscalability.com/google-architecture)
* [MySpace](http://highscalability.com/myspace-architecture)
* [Amazon](http://highscalability.com/amazon-architecture)
* [Twitter](http://highscalability.com/scaling-twitter-making-twitter-10000-percent-faster)

A pesar del abundante evidencia de que la normalización rara vez se escala, parece que muchos ingenieros de software se aferrarán desconfiadamente a la normalización total de la base de datos por principio solo, mucho después de que haya dejado de tener sentido.

Pat Helland señala que las personas normalizan porque sus profesores se lo pidieron. Aunque Atwood afirma que es un poco más pragmático; cree que se debería normalizar cuando los datos te dicen que:

* La normalización tiene sentido para tu equipo.
* La normalización proporciona un mejor rendimiento. (Estás midiendo automáticamente todas las consultas que fluyen a través de tu software, ¿verdad?)
* La normalización evita una cantidad onerosa de duplicaciones o evita el riesgo de problemas de sincronización a los que su dominio problemático o los usuarios son particularmente sensibles.
* La normalización le permite escribir consultas y códigos más simples.

Finalmente, el autor aconseja que nunca, nunca se debería normalizar una base de datos por un vago sentido del deber hacia los fantasmas de Boyce-Codd. La normalización a menudo crea tantos problemas como resuelve. No se debe temer el espectro de la desnormalización. Los datos duplicados y los problemas de sincronización a menudo son exagerados y relativamente fáciles de solucionar con trabajos. Los discos y la memoria son baratos y son cada vez más económicos cada nanosegundo. Mida el rendimiento en su sistema y decida por usted mismo qué funciona, sin predisposiciones ni sesgos.

Como dice el viejo refrán, normalízate hasta que duela, desnormaliza hasta que funcione.

[**10 Reasons To Consider A Multi-Model Database**](http://highscalability.com/blog/2015/3/4/10-reasons-to-consider-a-multi-model-database.html)

**(autor:** [**Nikhil Palekar**](http://blog.foundationdb.com/author/nikhil-palekar)***)***

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

**Base De Datos De Modelos Múltiples**

La reproducción de bases de datos NoSQL es una respuesta a las necesidades de las aplicaciones modernas. Aún así, no todos los datos pueden ser ajustados a un modelo NoSQL particular, razón por la cual existen tantas opciones de bases de datos diferentes en el mercado. Como resultado, las organizaciones ahora se enfrentan a una gran cantidad de bases de datos dentro de su infraestructura.

Los beneficios para las organizaciones son amplios, pero algunos de los beneficios más importantes incluyen:

**1. Consolidación**

En el espacio NoSQL en particular, los ingenieros se enfrentan a muchas opciones cuando deciden cómo modelar y almacenar datos.

**2. Escalado De Rendimiento**

A medida que crece el uso de una aplicación, la necesidad de rendimiento de la base de datos también aumenta.

**3. Complejidad Operacional**

Los entornos fragmentados causados ​​por ejecutar diferentes bases de datos aumentan la complejidad tanto de las operaciones como del desarrollo. El objetivo de la persistencia políglota es utilizar el mejor componente para el trabajo, pero en la práctica significa que puede terminar con múltiples bases de datos, cada una con sus propios requisitos operativos y de almacenamiento

**4. Flexibilidad**

A menudo es incómodo e ineficaz meter muchos datos en un solo modelo de datos. Un enfoque multimodelo implica mapear múltiples modelos de datos en un único motor de almacenamiento subyacente que puede admitir diferentes casos de uso y aplicaciones.

**5. Confiabilidad**

La confiabilidad de la base de datos también es un problema cuando se ejecutan múltiples bases de datos, ya que cada sistema de base de datos podría ser un único punto de falla para el sistema y la aplicación de mayor tamaño.

**6. Consistencia De Los Datos**

Sin la funcionalidad de transacciones de alto nivel integrada en su aplicación, no hay soporte para transacciones en diferentes sistemas de bases de datos. En consecuencia, no hay una buena manera de mantener la coherencia entre los diferentes modelos. Supongamos que su aplicación recibe una secuencia de datos sobre la actividad del usuario y decide almacenar elementos de datos relacionados en una estructura de series de tiempo, formato de gráfico y formato de documento.

**7. Tolerancia A Fallas**

Asegurar que un sistema con muchos componentes de cualquier tipo sea tolerante a fallas no es una tarea fácil, por decir lo menos. Integrar múltiples sistemas que fueron diseñados para ejecutarse de forma independiente, de modo que proporcionan tolerancia a fallas en todo el sistema en conjunto, impone costos de ingeniería y operativos significativos. Desafortunadamente, las implementaciones de sistemas heterogéneos requieren que su equipo tenga experiencia con cada componente para que el sistema en general funcione correctamente. Debido a que cada sistema es diferente y tiene diferentes requisitos, sin embargo, este enfoque lleva mucho tiempo y es costoso. E incluso entonces, la tolerancia a fallas de todo su sistema depende entonces del subsistema más débil en el back-end.

**8. Costo**

Al usar más, los distintos sistemas de bases de datos aumentan los costos según el hardware, el software y las necesidades operativas asociadas con cada sistema.

**9. Transacciones**

Los sistemas de bases de datos relacionales, generalmente implementados en una sola máquina, generalmente ofrecen fuertes garantías transaccionales para las operaciones de la base de datos, lo que permitió a las aplicaciones y desarrolladores de aplicaciones comprender con certeza el estado actual de la base de datos en un momento determinado. Sin embargo, es un desafío proporcionar transacciones entre varias máquinas, y casi todas las bases de datos NoSQL no ofrecen garantías transaccionales debido a sus diseños arquitectónicos.

**10. Mejores Aplicaciones**

Intentar ejecutar diferentes bases de datos para alimentar una aplicación puede ser una pesadilla operacional y de desarrollo. Por el contrario, una aplicación que es compatible con una base de datos de modelos múltiples obtiene los beneficios de escalabilidad, tolerancia a fallas y en un sistema bien diseñado, alto rendimiento integrado en el producto.