**Redis:**

Es un almacén de estructura de datos de valores en memoria rápido y de código abierto basado en C que no pertenece a la familia de los lenguajes de consulta estructurada (SQL) por lo cual no almacena la información en una estructura bien definida como lo son tablas, columnas y filas, en redis usan listas,conjuntos, bitmaps entre otros que SQL no maneja de una manera óptima, además almacena cadenas como python o java que puede contener cualquier tipo de dato con un tamaño máximo de 520mb.

Es comúnmente usado por AIRBNB,UBER,INSTAGRAM,PRINTERES,REDIDT, TWITTER (por mencionar las mas conocidas) entre otras

**Tarjetas de crédito y Rappi:**

El principal problema fue que rappi estaba dejando expuesta la información sensible del usuario en el código de la página web propia y una persona que tenga conocimientos en pagina web y hacking podría acceder fácilmente a dicha información y para ello debían cumplir con el PCI DSS, el cual es una herramienta para evaluar el cumplimiento de las normas de seguridad de las tarjetas de crédito y que la trazabilidad entre cliente y proveedor no se vean afectadas, para ello tiene varias políticas las cuales son:

Que si se quiere que el usuario ingrese información de pago en un contenedor HTML entonces que use un contendor especial de pagos que ya ha sido creado.

Generar la información de pago del cliente desde el navegador usando javascript encriptado

Por ultimo que use APIs ya probadas y funcionales para ello

Por lo cual es recomendable usar los protocolos o características indicados en el PCI DSS para evitar fraudes a la hora de realizar proyectos con tarjetas de crédito

**10 razones para considerar una base de datos multimodelo:**

1. Consolidación: como las bases de datos multimodelo soportan diferentes tipos de datos estos se consolidan en una plataforma, minimizando así su mantenimiento y dando una estructura mas cómoda
2. Escala de rendimiento: en estos momentos con la evolución del software en especial de las aplicaciones, se debe tener en cuenta que una base de datos no debe solo crecer verticalmente sino también horizontalmente para hacer que el rendimiento de las aplicaciones mejore
3. Complejidad operacional: cuando un sistema esta separado en ambientes distintos se incrementa la complejidad de las operaciones a la hora de usar una sola base de datos, por lo cual una base de datos multinivel ayudaría a mejorar la integración de los datos cada uno con sus requisitos operativos
4. Flexibilidad: como en el caso anterior la flexibilidad de que los datos tengan cada uno sus requisitos operativos hacen que esta base de datos tenga mas uso, aplicaciones y enfoque que se puede admitir para el almacenamiento de datos
5. Fiabilidad: es uno de los términos mas importantes a la hora de tener una base de datos, puesto que hay que garantizar su correcto funcionamiento en cada momento que sea necesario, por lo cual una caída podría tomar horas que acarrearían problemas monetarios y de experiencia de usuario
6. Consistencia de datos: sin una buena funcionalidad integrada a la aplicación, no habría una buena transaccionalidad en la base de datos por lo cual los datos se volverían confusos, por ende un único backend que admite varios modelos de requisitos , puede hacer que la toma de datos sea mas simple.
7. Tolerancia a fallos: uno de los problemas mas a tener en cuenta es este y mas cuando se tiene un sistema tiene requisitos distintos y es independiente, por lo cual esta tolerancia al fallo depende mas de los sistemas derivados de otros en el backend
8. Costos: lo mas problemático acá es el mantenimiento del sistema puesto que esto conlleva a que se retrasen tareas por hacer mantenimientos ya sean actualizaciones, parches u otras modificaciones para corregir errores detectados
9. Transacciones: en los sistemas multimodelo cada una de sus partes conoce como se almacenan los datos por ello la trazabilidad entre diferentes dispositivos traen bueno beneficios
10. Mejores aplicaciones: una aplicación multimodelo trae todos los beneficios antes expuestos , por lo cual se necesitaría menos lógica para el desarrollo de aplicativos por ende esto llevaría a que se mejorar a en la creación de aplicaciones distribuidas

[**Maybe Normalizing Isn't Normal**](https://blog.codinghorror.com/maybe-normalizing-isnt-normal/)**:**

en este caso se toca el tema de que normalizar o no es un tema controversial puesto que muchas veces hacer consultas de datos requieren muchos joins para traer algo sencillo que se podría hacer una tabla no normalizada, pero también en una tabla no normalizada a la hora de extraer información daría mucha irrelevante , ellos dicen que se debe trabajar según el entorno y que siempre no es necesario normalizar sino hacer que las tareas funcionen eficaz y eficientemente

[**Who Needs Stored Procedures, Anyways?**](https://blog.codinghorror.com/who-needs-stored-procedures-anyways/) **:**

En este articulo se toca el uso de los procedimientos almacenados y realmente si son útiles en la practica real, lo cual el autor dice que NO, porque las nuevas tecnologías (a partir del sql server 7.0) se dice que esto dependería mas del internet que se tiene, la traducción de del texto escrito (query) a la base de datos y los cálculos necesarios para traer la información , por lo cual ya no estarían siendo funcionales los procedimientos almacenados puesto que estos no traen información de errores, tienen una programación robusta, esconden lo que realmente está haciendo en el proceso.