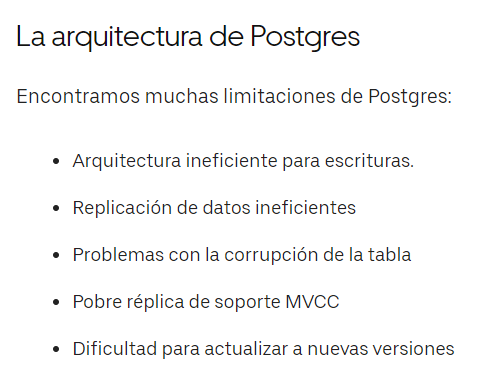
# **Por qué Uber Engineering cambió de Postgres a MySQL**

La arquitectura inicial de Uber consistía en una aplicación de backend monolítica escrita en Python que usaba [Postgres](http://www.postgresql.org/" \t "_blank) para la persistencia de datos. Desde entonces, la arquitectura de Uber ha cambiado significativamente, a un modelo de [microservicios](https://eng.uber.com/soa/) y nuevas plataformas de datos. Específicamente, en muchos de los casos en los que anteriormente [usábamos](https://eng.uber.com/schemaless-part-one/) Postgres, ahora usamos [Schemaless](https://eng.uber.com/schemaless-part-one/" \t "_blank) , una novedosa capa de fragmentación de base de datos construida sobre MySQL. En este artículo, exploraremos algunos de los inconvenientes que encontramos con Postgres y explicaremos la decisión de construir Schemaless y otros servicios backend sobre MySQL.

### **La arquitectura de MySQL**

Además de explicar algunas de las limitaciones de Postgres, también explicamos por qué MySQL es una herramienta importante para los nuevos proyectos de almacenamiento de Uber Engineering, como Schemaless. En muchos casos, encontramos a MySQL más favorable para nuestros usos. Para comprender las diferencias, examinamos la arquitectura de MySQL y su contraste con la de Postgres. Analizamos específicamente cómo funciona MySQL con el [motor de almacenamiento InnoDB](http://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html) . No solo usamos InnoDB en Uber; Es quizás el motor de almacenamiento MySQL más popular.



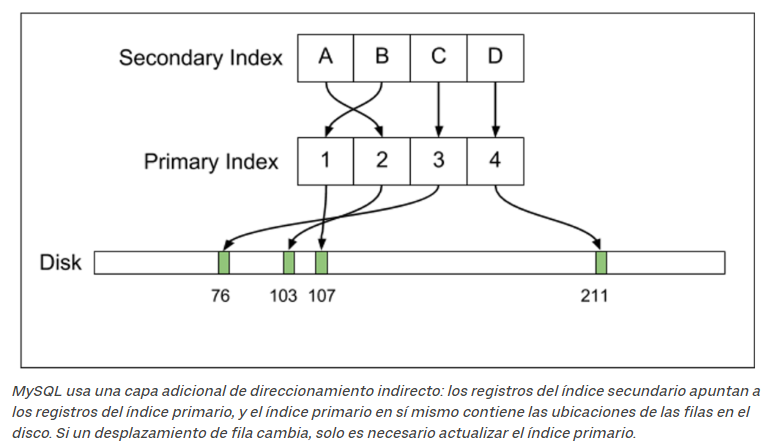
* **Arquitectura ineficiente para escrituras.**

El primer problema con el diseño de Postgres se conoce en otros contextos como [amplificación de escritura](https://en.wikipedia.org/wiki/Write_amplification) . Normalmente, la amplificación de escritura se refiere a un problema con la escritura de datos en discos SSD: una pequeña actualización lógica (por ejemplo, escribir unos pocos bytes) se convierte en una actualización mucho más grande y costosa cuando se traduce a la capa física.

La amplificación de escritura ( WA ) es un fenómeno indeseable asociado con [la memoria flash](https://en.wikipedia.org/wiki/Flash_memory) y las [unidades de estado sólido](https://en.wikipedia.org/wiki/Solid-state_drive) (SSD) donde la cantidad real de información escrita físicamente en los medios de almacenamiento es un múltiplo de la cantidad lógica que se desea escribir.

**MYSQL**

La diferencia arquitectónica más importante es que mientras Postgres mapea directamente los registros de índice a las ubicaciones en el disco, InnoDB mantiene una estructura secundaria. En lugar de mantener un puntero a la ubicación de la fila en el disco (como lo hace ctid en Postgres), los registros de índice secundario de InnoDB mantienen un puntero al valor de la clave principal.



* **Replicación de datos ineficientes**

Este problema fue extremadamente molesto por varias razones. Para empezar, no podríamos saber fácilmente a cuántas filas afectó este problema. Los resultados duplicados devueltos de la base de datos hicieron que la lógica de la aplicación fallara en varios casos. Terminamos agregando instrucciones de programación defensivas para detectar la situación de las tablas que se sabe tienen este problema. Debido a que el error afectó a todos los servidores, las filas dañadas eran diferentes en diferentes instancias de réplica, lo que significa que en una réplica la fila *X* podría ser mala y la fila *Y* sería buena, pero en otra réplica la fila *X* podría ser buena y la fila *Y* podría ser malo. De hecho, no estábamos seguros de la cantidad de réplicas con datos dañados y de si el problema había afectado al maestro.

**MYSQL** soporta múltiples [modos de replicación diferentes](https://dev.mysql.com/doc/refman/5.7/en/replication-formats.html) :

* La replicación basada en sentencias replica sentencias de SQL lógicas (p. Ej., Replicaría literalmente sentencias literales como: ACTUALIZAR a los usuarios SET birth\_year = 770 WHERE id = 4 )
* La replicación basada en filas replica registros de fila alterados
* La replicación mixta mezcla estos dos modos.

La arquitectura de replicación de MySQL significa que si los errores causan la corrupción de la tabla, es poco probable que el problema cause un error catastrófico. La replicación ocurre en la capa lógica, por lo que una operación como reequilibrar un [árbol B](https://en.wikipedia.org/wiki/B-tree) nunca puede causar que un índice se corrompa. Un problema típico de la replicación de MySQL es el caso de una declaración que se omite (o, con menos frecuencia, se aplica dos veces). Esto puede hacer que los datos falten o no sean válidos, pero no causará una interrupción de la base de datos.

* **Problemas con la corrupción de la tabla**

Durante una promoción de la base de datos maestra de rutina para aumentar la capacidad de la base de datos, se encontraron un error en Postgres 9.2. Las réplicas siguieron los [cambios de línea de tiempo incorrectamente](http://www.postgresql.org/docs/9.2/static/continuous-archiving.html) , lo que provocó que algunos de ellos aplicaran [incorrectamente](http://www.postgresql.org/docs/9.2/static/continuous-archiving.html) algunos registros WAL. Debido a este error, algunos registros que deberían haberse marcado como inactivos por el mecanismo de control de versiones no se marcaron como inactivos.

Comenzaron con Postgres 9.1 y completaron con éxito el proceso de actualización para pasar a Postgres 9.2. Sin embargo, el proceso tomó tantas horas que no pudieron hacer el proceso nuevamente. Cuando se publicó Postgres 9.3, el crecimiento de Uber aumentó sustancialmente nuestro conjunto de datos, por lo que la actualización habría sido aún más larga. Por este motivo, las instancias heredadas de Postgres ejecutan Postgres 9.2 hasta el día de hoy, aunque la versión actual de Postgres GA es 9.5.

* **Pobre réplica de soporte MVCC**

Postgres no tiene verdadera réplica de soporte MVCC. El hecho de que las réplicas apliquen las actualizaciones de WAL [hace que tengan una copia de los datos en el disco idéntica a la maestra](http://blog.2ndquadrant.com/tradeoffs_in_hot_standby_deplo/) en un momento dado. Este diseño plantea un problema para Uber.

Postgres necesita mantener una copia de las versiones anteriores de la fila para MVCC. Si una réplica de transmisión tiene una transacción abierta, las actualizaciones de la base de datos se bloquean si afectan a las filas que la transacción mantiene abiertas. En esta situación, Postgres detiene el subproceso de la aplicación WAL hasta que la transacción haya finalizado. Esto es problemático si la transacción lleva mucho tiempo, ya que la réplica puede retrasarse considerablemente con respecto al maestro. Por lo tanto, Postgres aplica un tiempo de espera en tales situaciones: si una transacción bloquea la aplicación WAL por un [tiempo determinado](https://www.postgresql.org/docs/9.2/static/hot-standby.html#HOT-STANDBY-CONFLICT) , Postgres mata esa transacción.

**MYSQL**

La forma en que funciona cada flujo de replicación también tiene una consecuencia importante sobre cómo funciona MVCC con las réplicas. Dado que la secuencia de replicación de MySQL tiene actualizaciones lógicas, las réplicas pueden tener una verdadera semántica de MVCC; por lo tanto, las consultas de lectura en las réplicas no bloquearán la secuencia de replicación. En contraste, la secuencia WAL de Postgres contiene cambios físicos en el disco, por lo que las réplicas de Postgres no pueden aplicar actualizaciones de replicación que entren en conflicto con las consultas de lectura, por lo que no pueden implementar MVCC.

* **Dificultad para actualizar a nuevas versiones**

Debido a que los registros de replicación funcionan a nivel físico, no es posible replicar datos entre diferentes versiones de disponibilidad general de Postgres. Una base de datos maestra que ejecuta Postgres 9.3 no se puede replicar en una réplica que ejecuta Postgres 9.2, ni tampoco se puede replicar una maestra que ejecuta 9.2 en una réplica que ejecuta Postgres 9.3.

**MYSQL**

Finalmente, la arquitectura de replicación de MySQL hace que sea fácil replicar entre diferentes versiones de MySQL. MySQL solo incrementa su versión si el formato de replicación cambia, lo cual es inusual entre varias versiones de MySQL. El formato de replicación lógica de MySQL también significa que los cambios en el disco en la capa del motor de almacenamiento no afectan el formato de replicación. La forma típica de realizar una actualización de MySQL es aplicar la actualización a una réplica a la vez, y una vez que actualice todas las réplicas, promocione una de ellas para convertirse en el nuevo maestro. Esto se puede hacer con casi cero tiempos de inactividad, y simplifica el mantenimiento de MySQL actualizado.

#### La piscina de búfer

Primero, el almacenamiento en caché funciona de manera diferente en las dos bases de datos. Postgres asigna algo de memoria para cachés internos, pero estos cachés son típicamente pequeños en comparación con la cantidad total de memoria en una máquina. Para aumentar el rendimiento, Postgres permite que el kernel almacene automáticamente en caché los datos de disco accedidos recientemente a través del [caché de página](https://en.wikipedia.org/wiki/Page_cache) . Por ejemplo, nuestras réplicas más grandes de Postgres tienen 768 GB de memoria disponible, pero solo alrededor de 25 GB de esa memoria es en realidad una [memoria RSS](https://en.wikipedia.org/wiki/Resident_set_size) en la que los procesos de Postgres han fallado. Esto deja más de 700 GB de memoria libre al caché de la página de Linux.

Manejo de la conexión

MySQL implementa conexiones concurrentes al generar un hilo por conexión.  Sin embargo, Postgres usa un diseño de proceso por conexión. Esto es significativamente más costoso que un diseño de conexión por conexión por varias razones.

### **Conclusión del autor**

Postgres nos sirvió bien en los primeros días de Uber, pero nos encontramos con problemas importantes en la escala de Postgres con nuestro crecimiento. Hoy en día, tenemos algunas instancias de Postgres heredadas, pero la mayor parte de nuestras bases de datos están construidas sobre MySQL (generalmente utilizando nuestra capa [Schemaless](https://eng.uber.com/schemaless-part-one/" \t "_blank) ) o, en algunos casos especializados, bases de datos NoSQL como Cassandra. En general, estamos bastante contentos con MySQL, y es posible que en el futuro tengamos más artículos de blog que expliquen algunos de sus usos más avanzados en Uber.

## [**Netflix: ¿Qué Sucede Cuando Presionas Play?**](http://highscalability.com/blog/2017/12/11/netflix-what-happens-when-you-press-play.html)

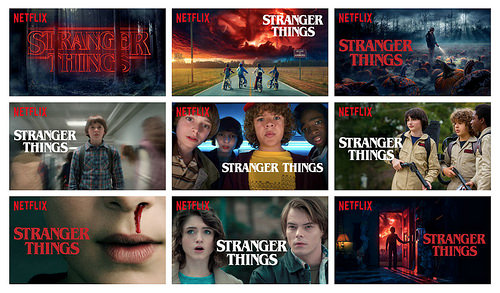
Para ver por qué echemos un vistazo a algunas estadísticas impresionantes de Netflix para 2017.

* Netflix tiene más de 110 millones de suscriptores.
* Netflix opera en más de 200 países.
* Netflix tiene casi $ 3 mil millones en ingresos por trimestre.
* Netflix agrega más de 5 millones de nuevos suscriptores por trimestre.
* Netflix reproduce más de 1 billón de horas de video cada semana. Como comparación, YouTube transmite 1 billón de horas de video *todos los días,* mientras que Facebook transmite 110 millones de horas de video todos los días.
* Netflix jugó 250 millones de horas de video en un solo día en 2017.
* Netflix representa más del 37% del tráfico pico de Internet en los Estados Unidos.
* Netflix planea gastar $ 7 mil millones en contenido nuevo en 2018.

**Netflix opera en dos nubes: AWS y Open Connect.**

**Las tres partes de Netflix:**

|  |  |  |
| --- | --- | --- |
| **Cliente**  iPhone, TV, Xbox, teléfono Android, tableta, etc.  Netflix Controla El Cliente  Netflix maneja los fallos con gracia porque controla el cliente en todos los dispositivos que ejecutan Netflix. | **Backend**  **Controlado por AWS(Amazon Web Services)** **En 2008 Netflix Comenzó A Mudarse A AWS**Tenían sus propios servidores,era costoso,**Una Interrupción Del Servicio Hizo Que Netflix Se Trasladara A AWS,** porque quería una infraestructura más confiable AWS hace todo el trabajo pesado indiferenciado para Netflix  Netflix tardó más de ocho años en completar el proceso de pasar de sus propios centros de datos a AWS  Netflix opera en tres regiones de AWS: una en el norte de Virginia, una en Portland, Oregon, y una en Dublín, Irlanda. Dentro de cada región, Netflix opera en tres zonas de disponibilidad diferentes.  Esto incluye computación escalable, almacenamiento escalable, lógica empresarial, bases de datos distribuidas escalables (DynamoDB y Cassandra para sus bases de datos distribuidas), análisis y procesamiento de big data, recomendaciones, transcodificación y cientos de otras funciones. **Netflix Ahorra Dinero En AWS** | **Red de entrega de contenido (CDN).**  su propio CDN creado específicamente …Open Connect almacena el video de Netflix en diferentes lugares del mundo. Cuando presiona reproducir las transmisiones de video de Open Connect, en su dispositivo  Los dispositivos Open Connect son servidores especialmente diseñados que se centran en baja potencia y alta densidad de almacenamiento  Menos costoso, mejor calidad, más escalable  Muchos servidores conectados al tiempo  Netflix ofrece una gran cantidad de tráfico de video desde miles de servidores en más de 1,000 ubicaciones en todo el mundo  Lo que Netflix ha hecho con Open Connect es colocar sus clusters OCA dentro de la red de los ISP. Eso significa que si veo un video de Netflix estaré hablando con un OCA en la red de Comcast. Todo mi tráfico de video está en la red de Comcast; nunca llega a internet. |
| Netflix ha logrado una integración vertical completa.  Netflix ya consume más del 37% del tráfico de Internet en los Estados Unidos | | |



Es posible que se sorprenda al saber que la imagen que se muestra para cada video se selecciona específicamente para usted. No todos ven la misma imagen.

impulsado por datos . Netflix es conocido por ser una empresa basada en datos. Los datos se recopilan, en este caso, el número de vistas asociadas con cada imagen, y se utilizan para tomar las mejores decisiones posibles, en este caso, qué imagen de encabezado se debe seleccionar.

Recuerde, Netflix registra y cuenta todo lo que hace en su sitio. Saben qué tipo de películas te gustan más, qué actores te gustan más, y así sucesivamente.

¿Cómo decide Netflix? Utilizando el aprendizaje automático.

Eso es parte del procesamiento y análisis de big data del que acabamos de hablar. Netflix mira sus datos y predice lo que te gustará. De hecho, todo lo que ves en una pantalla de Netflix fue elegido específicamente para ti mediante el aprendizaje automático.

Antes de poder ver un video en el dispositivo favorito de su elección, Netflix debe convertir el video a un formato que funcione mejor para su dispositivo. Este proceso se llama transcodificación o codificación .

Netflix codifica todos sus videos en AWS en hasta 300,000 CPU a la vez. ¡Eso es más grande que la mayoría de las súper computadoras!

¿Quién envía el video a Netflix? Casas de producción y estudios. Netflix llama a este medio fuente de video . El nuevo video se entrega al equipo de operaciones de contenido para su procesamiento.

Antes de que pueda ver un video, Netflix lo somete a un riguroso proceso de varios pasos.

Necesitan muchos servidores para procesar estos enormes archivos de video en paralelo

Netflix no opera su propia red; Tampoco opera sus propios centros de datos. En su lugar, los proveedores de servicios de Internet (ISP) acuerdan poner OCA en sus centros de datos. Las OCA se ofrecen de forma gratuita a los ISP para integrarlas en sus redes. Netflix también coloca OCA en o cerca de las ubicaciones de intercambio de Internet (IXP).

Servidores - Hay varios tipos diferentes de OCA para diferentes propósitos. Hay grandes OCA que pueden almacenar todo el catálogo de videos de Netflix. Hay OCA más pequeños que pueden almacenar solo una parte del catálogo de videos de Netflix. Los OCA más pequeños se llenan de video todos los días, durante las horas de menor actividad, mediante un proceso que Netflix denomina caché proactivo los OCA utilizan el sistema operativo FreeBSD

En total, Netflix soporta 2200 dispositivos diferentes. Cada dispositivo tiene un formato de video que se ve mejor en ese dispositivo en particular. Si está viendo Netflix en un iPhone, verá un video que le brinda la mejor experiencia de visualización en el iPhone.

Para The Crown, Netflix almacena alrededor de 1.200 archivos.

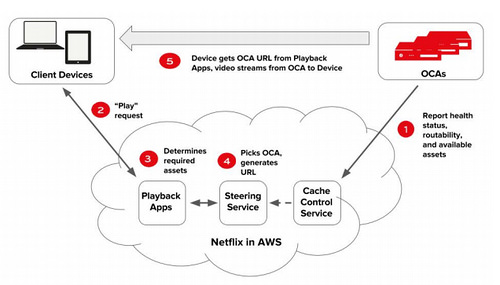
Netflix ha probado tres estrategias diferentes de transmisión de video: su propio CDN pequeño; CDN de terceros; y Open Connect.

La idea detrás de un CDN es simple: coloque el video lo más cerca posible de los usuarios al distribuir computadoras en todo el mundo. Cuando un usuario quiera ver un video, busque la computadora más cercana con el video y transmítalo al dispositivo desde allí.

Los mayores beneficios de un CDN son la velocidad y la fiabilidad.

## **El Video Se Guarda En Caché Proactivamente En Las OCA Todos Los Días**

En todas partes del mundo, Netflix, sabe con un alto grado de precisión lo que a sus miembros les gusta ver y cuándo les gusta verlo.



* Netflix se puede dividir en tres partes: el backend, el cliente y el CDN.
* Todas las solicitudes de los clientes de Netflix se manejan en AWS.
* Todo el video se transmite desde un dispositivo Open Connect cercano (OCA) en el Open Connect CDN.
* Netflix opera en tres regiones de AWS y generalmente puede manejar una falla en cualquier región sin que los miembros lo noten.
* Netflix transforma el nuevo contenido de video en muchos formatos diferentes, por lo que se puede seleccionar el mejor formato para ver según el tipo de dispositivo, la calidad de la red, la ubicación geográfica y el plan de suscripción del miembro.
* Todos los días, a través de Open Connect, Netflix distribuye video en todo el mundo, según lo que predicen los miembros en cada ubicación que querrán ver.

# **Fallas graves en la seguridad de tarjetas de crédito y credenciales en Rappi**

Todos los comercios que acepten tarjetas de crédito, sin importar su tamaño, deberían cumplir el estándar [PCI DSS](https://www.pcisecuritystandards.org/pci_security/)

De acuerdo con el [blog de Rappi](https://blog.rappi.com/rappi-le-apuesta-cerrar-el-ano-con-10-000-rappitenderos/) solo en Bogotá se ejecutan 10.000 transacciones al día. Al proyectar tendríamos casi 4 millones anuales solo en Bogotá. De acuerdo con esto Rappi debería tener mínimo un nivel PCI 2 o incluso superior.

los datos de las tarjetas de crédito se transfieren desde el computador o móvil del usuario a los servidores del procesador de pagos. Nunca pasan por el comercio.

Rappi, suponemos, con el fin de mejorar la experiencia del usuario, optó por no requerir los datos de la tarjeta en cada compra. Esto significa que no solo son recolectados en un formulario sino que están almacenados de alguna forma. Esto requiere mucho cuidado en la implementación ya que se incrementa la exposición de los datos .

Al momento de nuestra revisión Rappi usaba los servicios de [Spreedly](https://www.spreedly.com/pci/" \t "_blank) ; un servicio reputado de procesamiento de pagos y almacenamiento con los más altos estándares PCI nivel 1.

### **¿ Cuáles eran las fallas graves de Rappi\* ?**

Rappi, al presentar el formulario de recolección de datos de las tarjetas, opta por incluírlo desde sus servidores. Aunque los datos fueran enviados directamente al procesador de pagos esto incrementa la exposición de los datos de las tarjetas y les pone mayores exigencias en los cuestionarios de PCI, al menos SAQ A-EP. Rappi [no usó los mecanismos recomendados](https://docs.spreedly.com/guides/adding-payment-methods/) por el procesador de pagos para recolectar la información de las tarjetas y de esta forma tener menores exigencias por exposición de la información. Aunque esto es válido **no conocemos nada acerca de sus certificaciones PCI.**

**Verificamos mediante un servidor en cada país (proxy), revisando el código, que Rappi incluía las credenciales del mismo compartimento seguro en varios países en los que presta servicio y que no había diferencias notables en el archivo de código (Argentina, México, Brasil, Colombia,Chile)**

En un ataque más sofisticado que obtener los datos, y debido a la arquitectura errada de Rappi al momento de la revisión, un atacante pudo ejecutar otro tipo de ataque sobre las tarjetas de crédito\*. La aplicación web de Rappi luego de obtener del almacenamiento seguro el identificador (token) lo enviaba a sus servidores desde el navegador (ver flujo arriba) para ser asociado. Un atacante con un identificador (token) válido podría asociar el identificador (token) que representa la tarjeta de crédito de un tercero y proceder a realizar transacciones desde su cuenta u otra, con cargo a la tarjeta de esta otra persona. Aunque los identificadores (tokens) están diseñados para no ser adivinados fácilmente, obtener identificadores válidos no le sería difícil ya que el código de la página web de Rappi incluía las credenciales del compartimento seguro, de donde se pueden obtener tanto identificadores válidos, así como datos parciales de la tarjeta, datos personales y realizar operaciones.

### **¿ Cómo solucionó Rappi el reporte\*?**

Desde el reporte inicial Rappi tardó **61 días**en solucionar la falla (ver Cronología). Cuarenta y dos (42 )días para eliminar las credenciales de un archivo javascript de su página web y 19 días más cuando los contactamos, de nuevo, a informarles que **a pesar de eliminar las credenciales no las habían revocado (cambiado)**. Este nos parece un paso lógico luego de estar expuestas varios meses (no sabemos cuánto tiempo antes de nuestro reporte) de forma totalmente pública.

A la fecha el formulario se incluye desde sus servidores y dominio y adicionalmente ahora **los datos de las tarjetas los envían y se procesan de alguna forma en sus servidores en el cloud de Amazon**. Esta decisión aumenta la exposición de los datos de las tarjetas de crédito (ya que**van directo a los servidores administrados por Rappi o Grability. A pesar de que usen el data center de Amazon la seguridad es un proceso interno bajo su administración\***) y aunque no se almacenen son cobijados por el **estándar más exigente** de PCI SAQ-D.

# **¿Por qué SQL está superando a NoSQL y qué significa esto para el futuro de los datos?**

En la última década, esto provocó que los desarrolladores de software desecharan el SQL como una reliquia que no podía escalar con estos crecientes volúmenes de datos, lo que llevó al aumento de NoSQL: MapReduce y Bigtable, Cassandra, MongoDB y más.

Sin embargo, hoy el SQL está resurgiendo. Todos los principales proveedores de la nube ahora ofrecen servicios populares de bases de datos relacionales administradas: por ejemplo, [Amazon RDS](https://aws.amazon.com/rds/) , [Google Cloud SQL](https://cloud.google.com/sql/docs/) , [Azure Database para PostgreSQL](https://azure.microsoft.com/en-us/services/postgresql/) (Azure se lanzó este año). En las propias palabras de Amazon, su producto de base de datos Aurora compatible con PostgreSQL y MySQL ha sido el " [servicio de más rápido crecimiento en la historia de AWS](http://www.businesswire.com/news/home/20161130006131/en/AWS-Extends-Amazon-Aurora-PostgreSQL-Compatibility)". Las interfaces SQL en la parte superior de Hadoop y Spark continúan prosperando. Y justo el mes pasado, [Kafka lanzó soporte de SQL](https://www.confluent.io/blog/ksql-open-source-streaming-sql-for-apache-kafka/) . Sus humildes autores mismos son desarrolladores de una nueva [base de datos de series de tiempo](https://github.com/timescale/timescaledb) que abarca totalmente SQL.

En este post, examinamos por qué el péndulo de hoy está cambiando a SQL y lo que esto significa para el futuro de la comunidad de análisis e ingeniería de datos.

SQL, que se introdujo por primera vez en el mundo en 1974. En las próximas décadas, el SQL demostraría ser inmensamente popular. Como las bases de datos relacionales como System R, Ingres, DB2, Oracle, SQL Server, PostgreSQL, MySQL (y más) se hicieron cargo de la industria del software, el SQL se convirtió en el lenguaje preeminente para interactuar con una base de datos, y se convirtió en la lengua franca para Ecosistema abarrotado y competitivo.

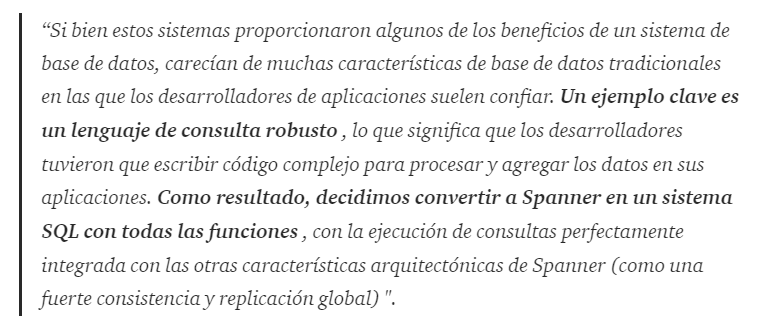
Hasta que apareció Internet

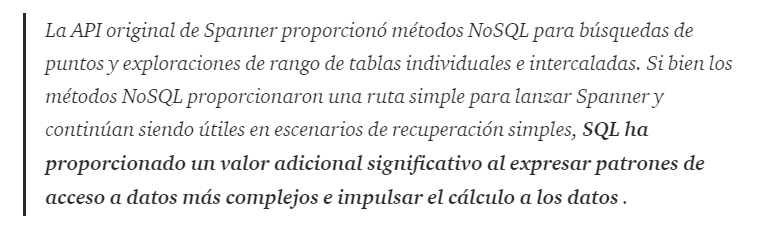
Y la comunidad de desarrolladores de software se comió NoSQL, abrazándolo mucho más ampliamente de lo que pretendían los autores originales de Google / Amazon. Es fácil entender por qué: NoSQL era nuevo y brillante; Prometió escala y poder; Parecía el camino rápido hacia el éxito de la ingeniería. Pero entonces empezaron a aparecer los problemas.

Los desarrolladores pronto descubrieron que no tener SQL era en realidad bastante limitante. Cada base de datos NoSQL ofrecía su propio lenguaje de consulta único, lo que significaba: más idiomas para aprender (y para enseñar a sus compañeros de trabajo); mayor dificultad para conectar estas bases de datos a aplicaciones, lo que lleva a toneladas de código de pegamento frágil; la falta de un ecosistema de terceros, que obliga a las empresas a desarrollar sus propias herramientas operativas y de visualización.

Un día nos dimos cuenta de que construir nuestro propio lenguaje de consulta no tenía sentido. Que la clave era abrazar SQL. Y esa fue una de las mejores decisiones de diseño que hemos tomado. Inmediatamente se abrió un mundo completamente nuevo.

Por ejemplo, Google comenzó a construir sobre Bigtable, pero luego descubrió que la falta de SQL creaba problemas





**SQL se ha convertido en la interfaz universal para el análisis de datos.**

Vivimos en una era donde los datos se están convirtiendo en "el recurso más valioso del mundo" ( [The Economist, mayo de 2017](https://www.economist.com/news/leaders/21721656-data-economy-demands-new-approach-antitrust-rules-worlds-most-valuable-resource" \t "_blank) )

¿SQL es perfecto? No, pero es el idioma que la mayoría de nosotros en la comunidad conocemos. Y aunque ya hay ingenieros trabajando en una interfaz más natural orientada al lenguaje, ¿a qué se conectarán esos sistemas? SQL