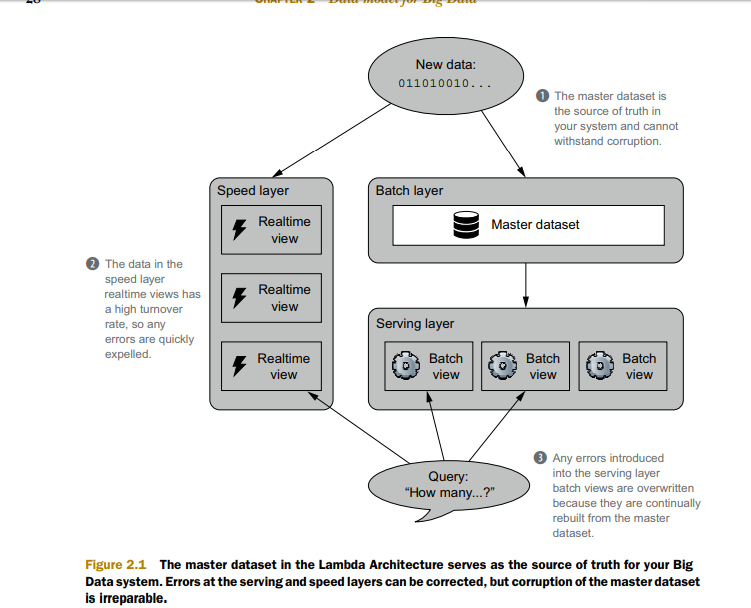
Articulo Big Data

Capitulo 2: Modelos de datos para Big Data

En el último capítulo usted vio lo que puede salir mal cuando se utilizan herramientas tradicionales para la construcción de sistemas de datos, y volvimos a primeros principios para obtener un mejor diseño. Se ha visto que cada sistema de datos puede ser formulado como funciones de datos de la computación, y has aprendido los conceptos básicos de la arquitectura Lambda, que proporciona una forma práctica para implementar una función arbitraria de datos arbitrarios en tiempo real.

El núcleo de la arquitectura Lambda es el conjunto de datos principal, que se destaca en la figura 2.1. El principal conjunto de datos es la fuente de la verdad en la arquitectura Lambda. Incluso si usted pierde todos sus datos de capa de servicio y velocidad capa datasets, podría reconstruir la aplicación del conjunto de datos principal. Esto es porque las vistas de lotes servidas por la capa de servicio se producen a través de funciones en el conjunto de datos principal, y puesto que la capa de velocidad se basa sólo en datos recientes, puede construir un sí mismo dentro de unas horas.

El principal conjunto de datos es la única parte de la arquitectura Lambda que absolutamente debe ser protegida de la corrupción. Máquinas de sobrecargadas, falta discos y cortes de energía podrían provocar errores, y errores humanos con los sistemas de datos dinámicos es un riesgo intrínseco y inevitable eventualidad. Cuidadosamente debe el Ingeniero principal ver el conjunto de datos para prevenir la corrupción en todos los casos, como tolerancia a errores es esencial para la salud de un sistema de datos de larga duración.



Hay dos componentes el principal conjunto de datos: los datos del modelo uso y cómo almacenar físicamente el principal conjunto de datos. Este capítulo es sobre el diseño de un modelo de datos para el conjunto de datos principal y las propiedades que debe tener un modelo de datos. Usted aprenderá sobre el almacenamiento físicamente de un conjunto de datos principal en el próximo capítulo. En este capítulo que vas a hacer lo siguiente:

■ aprender las propiedades clave de datos

■ ver cómo estas propiedades se mantienen en la

■ modelo basado en el hecho de examinar las ventajas del modelo hecho para el conjunto de datos principal

■ Express un modelo basado en el hecho de usar esquemas gráfico

Vamos a ser Gin con una discusión de los datos del término bastante general.

**2.1 Propiedades de la Data**

En consonancia con el enfoque aplicado del libro, a centro de nuestra discusión en torno a una aplicación de ejemplo. Supongamos que está diseñando la próxima gran red social — FaceSpace. Cuando un usuario nuevo, llamémosle Tom — se une a su sitio, comienza a invitar a sus amigos y familiares. ¿Información que debe guardar con respecto a las conexiones de Tom? Tienes un número de opciones, incluyendo las siguientes:

■ la secuencia del amigo de Tom y lista actual de unfriend eventos

■ Tom de número actual de amigos

■ Tom de amigos Figura 2.2 exhibe estas opciones y sus relaciones.

Este ejemplo ilustra la dependencia de la información. Tenga en cuenta que cada capa de información se puede derivar la anterior (el que está a su izquierda), pero de un proceso unidireccional. De la secuencia de eventos amigo y unfriend, puede determinar las otras cantidades. Del mismo modo, en la lista de amigos actuales, es imposible determinar si Tom estaba un amigo con Jerry, o red de Tom ha ido creciendo últimamente.

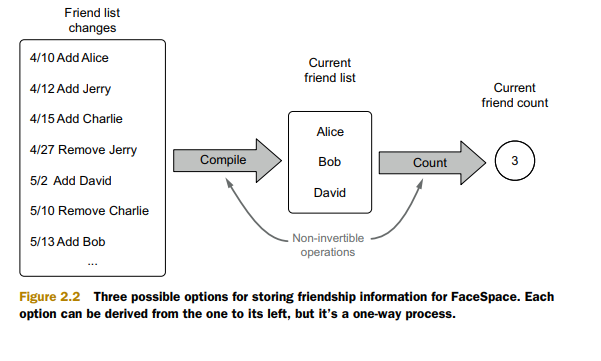
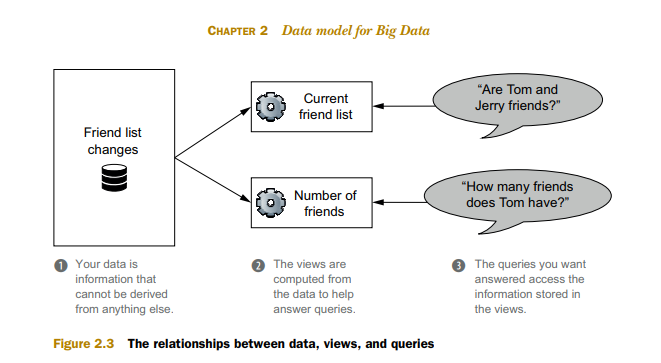
La noción de dependencia da forma a las definiciones de los términos que vamos a utilizar:

■ información es la colección general de conocimientos relevantes a su sistema de Big Data. Es sinónimo con el uso coloquial de los datos de la palabra.

■ los datos se refieren a la información que no puede derivar de cualquier otra cosa. Datos sirve como los axiomas del que se deriva todo lo demás.

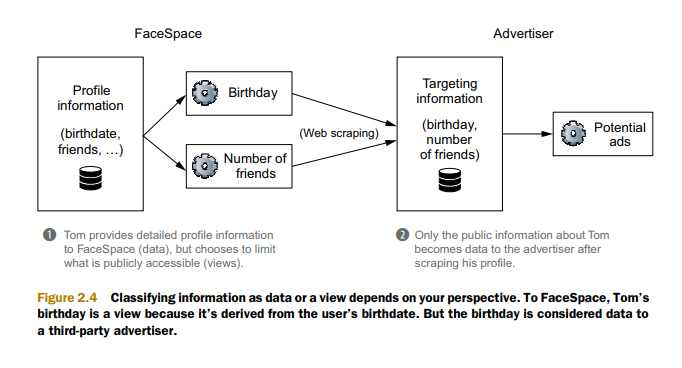
■ las consultas son preguntas que de sus datos. Por ejemplo, consultar su historial de transacciones financieras para determinar su saldo de cuenta bancaria.

■ las vistas son información que se ha derivado de su base de datos. Se construyen para ayudar a responder a determinados tipos de consultas.

Es importante observar que los datos de una persona pueden ser vista de otro. Supongamos que FaceSpace se convierte en un éxito monstruoso, y una empresa de publicidad crea un rastreador que raspa la información demográfica de los perfiles de usuario. FaceSpace tiene acceso completo a todas las informaciones Tom — por ejemplo, su fecha de nacimiento completa de 13 de marzo de 1984. Pero Tom es sensible sobre su edad, y él sólo hace su cumpleaños (13 de marzo) disponible en su perfil público. Su cumpleaños es una vista desde la perspectiva de FaceSpace porque se deriva de su fecha de nacimiento, pero sus datos al anunciante porque ellos tienen una limitada información sobre Tom. Esta relación se muestra en la figura 2.4.

Habiendo establecido un vocabulario compartido, ahora podemos introducir las propiedades clave de los datos: crudeza, inmutabilidad y perpetuidad (o la "eterna veracidad de datos").

 Fundamental para la comprensión de los sistemas de datos grandes es de su entendimiento de estos tres conceptos claves.

Si usted viene de un contexto relacional, esto podría ser confuso. Por lo general constantemente actualizar y resumir su información para reflejar el estado actual del mundo; no está preocupado con la inmutabilidad o perpetuidad. Pero ese enfoque limita las preguntas puede contestar con sus datos, y no desalentar enérgicamente errores y la corrupción. Mediante la aplicación de estas propiedades en el mundo del Big Data, puede lograr un sistema más robusto y tener capacidades de gran alcance.

A profundizar más en este tema como discutimos la crudeza de los datos.

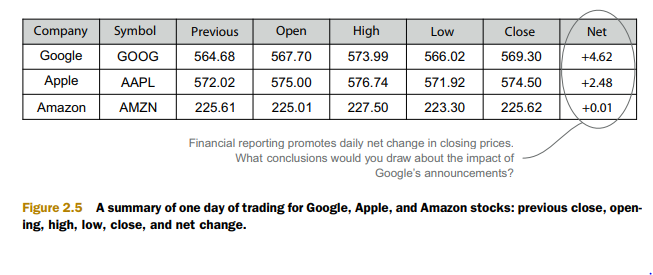
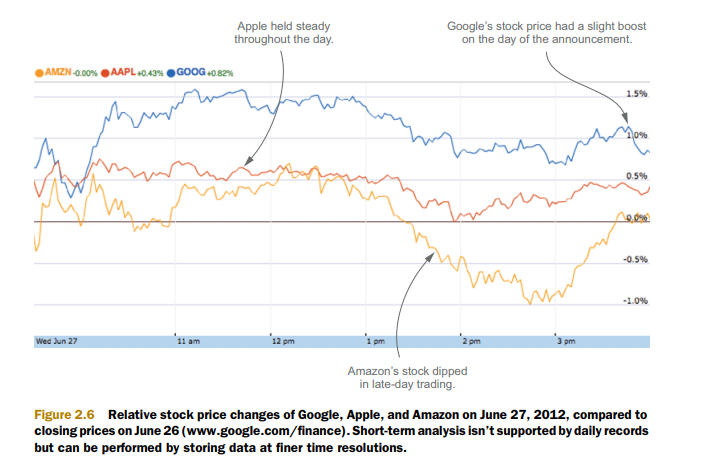
2.1.1 Data is raw

Un sistema de datos responde a las preguntas sobre la información que has adquirido en el pasado. Al diseñar su sistema de información grande, quiere ser capaz de responder a tantas preguntas como sea posible. En el ejemplo FaceSpace, sus datos FaceSpace están más valiosos que el anunciante porque puede deducir más información sobre Tom. Familiar, que llamaremos la crudeza de esta propiedad. Si se puede, desea almacenar los datos peor que usted puede conseguir sus manos en. El más crudo sus datos, las preguntas más que puedes hacer de él.

FaceSpace ejemplo ayuda a ilustrar el valor de crudeza, pero vamos a considerar otro ejemplo para ayudar a impulsar el punto de inicio. Bolsa de comercio es una fuente de información, con millones de acciones y miles de millones de dólares cambian de manos diariamente. Con muchos comercios tienen lugar, precios de las acciones históricamente se registran diariamente como precio de apertura, precio alto, precio bajo y precio de cierre. Pero esos bits de datos a menudo no proporcionan el panorama y potencialmente pueden sesgar la percepción de lo que sucedió. Por ejemplo, ver figura 2.5. Registra los datos del precio de las acciones de Google, Apple y Amazon en un día en que Google anunció nuevos productos dirigidos a sus competidores.

Este dato sugiere que Amazon puede no han sido afectado por el anuncio de Google, como su precio se movió solo un poco. También sugiere que el anuncio tenía ningún efecto sobre la manzana, o un efecto positivo.

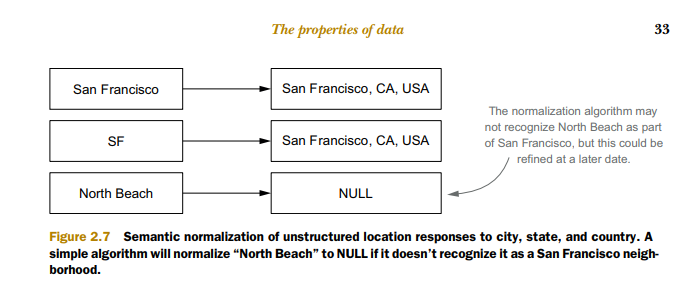
Pero si usted tiene acceso a los datos almacenados en una granularidad más fina del tiempo, se puede obtener una imagen más clara de los eventos que día y sonda más lejos en causa potencial y relaciones de efecto. Figura 2.6 muestra el minuto a minuto cambios relativos en los precios de las acciones de las tres empresas, lo cual sugiere que tanto Amazon como Apple efectivamente afectadas por el anuncio, Amazon más que Apple.

También tenga en cuenta que los datos adicionales pueden sugerir nuevas ideas que puede no haber considerado al examinar el Resumen diario original del precio de las acciones. Por ejemplo, los datos más detallados le pregunto si Amazon fue afectado más grandemente debido a los nuevos productos de Google compitan con Amazon en la tableta y el mercado de cloud computing.

Almacenamiento de datos es muy valioso porque rara vez sabe de antemano todas las preguntas respondió. Al mantener los datos peor posible, maximizar su capacidad para obtener nuevos conocimientos, resumiendo, sobrescribir o eliminar información limitaciones lo pueden decirle los datos. La compensación es datos más crudos normalmente produc más de lo mismo, a veces mucho más. Pero Big Data tecnologías están diseñadas para administrar petabytes y exabytes de datos. Específicamente, administrar el almacenamiento de sus datos de forma distribuida, escalable sosteniendo la posibilidad de consultar directamente los datos.

Aunque el concepto es sencillo, no siempre es claro qué información se debe almacenar los datos crudos. Le proporcionaremos un par de ejemplos para ayudar a guiarle en la toma de esta decisión.

DATOS no estructurados los datos es más crudo que NORMALIZÓ al decidir qué datos raws para almacenar, una zona borrosa común es la línea entre normalización análisis y semántica. La normalización semántica es el proceso de remodelación de información libre en una forma estructurada de datos. 

Por ejemplo, FaceSpace puede solicitar ubicación de Tom. Él puede de entrada nada de ese campo, tales como San Francisco, CA, SF, Playa Norte y así sucesivamente. Un algoritmo de normalización semántica trataría de igualar la entrada con un lugar conocido, véase la figura 2.7.

¿Si te encuentras con un formulario de datos como una cadena de ubicación no estructurada, debe almacenar la cadena no estructurada o la forma semánticamente normalizada? Sostenemos que es mejor almacenar la cadena no estructurada, debido a su algoritmo de normalización semántica puede mejorar con el tiempo. Si se guarda la cadena no estructurada, puede renormalize que los datos en un momento posterior cuando han mejorado sus algoritmos. En el ejemplo anterior, más adelante podrá adaptar el algoritmo para reconocer la playa del norte como un barrio de San Francisco, o puede que desee utilizar la información del barrio para otro propósito.

ALMACENAR DATOS NO ESTRUCTURADOS CUANDO... Como regla general, si el algoritmo para la extracción de los datos es sencilla y precisa, como la extracción de una edad de una página HTML, debe almacenar el resultado de ese algoritmo. Si el algoritmo está sujeto a cambios, debido a mejoras o ampliación de los requisitos, guarde la forma no estructurada de los datos.

MÁS INFORMACIÓN NO NECESARIAMENTE SIGNIFICA MÁS CRUDO DATOS :

es fácil presumir que más datos equivale a datos más crudos, pero eso no es siempre el caso. Vamos a decir que Tom es un blogger, y quiere añadir sus mensajes a su perfil de FaceSpace. ¿Qué exactamente debe almacenar una vez Tom proporciona la URL de su blog? Almacenar el texto puro de las entradas del blog sin duda es una posibilidad. Pero cualquier frases en cursiva, negrita o letra grande fueron deliberadamente enfatizados por Tom y podrían resultar útiles en el análisis del texto. Por ejemplo, puede utilizar esta información adicional para un índice para realizar búsquedas FaceSpace. Así podría afirmar que las entradas de texto anotado son una forma más cruda de los datos de cadenas de texto ASCII.

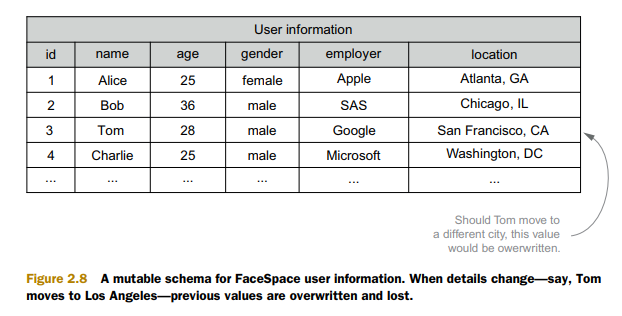
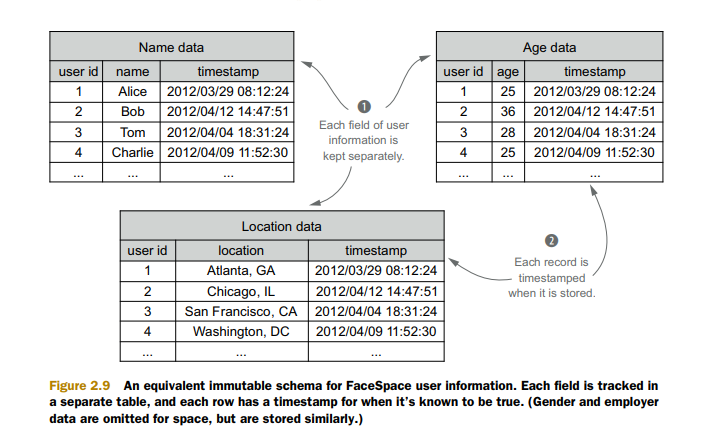
En el otro extremo del espectro, también se podría almacenar el código HTML completo del blog de Tom como sus datos. Mientras que es considerablemente más información en términos de bytes totales, el esquema de color, hojas de estilo y código JavaScript del sitio no pueden utilizarse para obtener información adicional acerca de Tom. Sólo sirven como contenedor para el contenido del sitio y no debería ser parte de tus datos.

2.1.2 Data is immutable

Datos inmutables pueden parecer un concepto extraño, si estás versado en bases de datos relacionales. Después de todo, en el mundo de base de datos relacional — y la mayoría otras bases de datos así: update es una de las operaciones fundamentales. Pero para inmutabilidad no actualizar o eliminar datos, añades sólo mas mediante el uso de un esquema inmutable para sistemas grandes de datos, se obtendrá dos ventajas vitales:

* Tolerancia a fallos humanos, esta es la ventaja más importante del modelo inmutable. Como ya comentamos en el capítulo 1, tolerancia a fallos humanos es una propiedad esencial de los sistemas de datos. Personas cometerán errores, y debe limitar el impacto de tales errores y tienen mecanismos para recuperarse de ellos. Con un modelo de datos mutables, un error puede provocar datos se perderán, porque realmente se reemplazan los valores en la base de datos. Con un modelo de datos inmutables, no hay datos se puede perder. Si es escritos datos incorrectos, aún existen unidades (calidad) de los datos anteriores. Corregir el sistema de datos es sólo una cuestión de eliminar las unidades de datos erróneos y recálculo de los puntos de vista construidos a partir de la base de datos principal.
* Simplicidad, los modelos de datos mutables implican que los datos deben ser indexados de alguna manera para que los objetos de datos específicos pueden ser obtenidos y actualizados. En cambio, con un modelo de datos inmutables basta la capacidad de añadir nuevas unidades de datos para el conjunto de datos principal. Esto no requiere un índice para los datos, que es una enorme simplificación. Como verá en el próximo capítulo, almacenar un conjunto de datos principal es tan simple como utilizar archivos planos.

Las ventajas de mantener sus datos inmutables se hacen evidentes cuando se comparan con un esquema mutable. Considerar el esquema básico de mutable que se muestra en la figura 2.8, que se podría utilizar para FaceSpace.

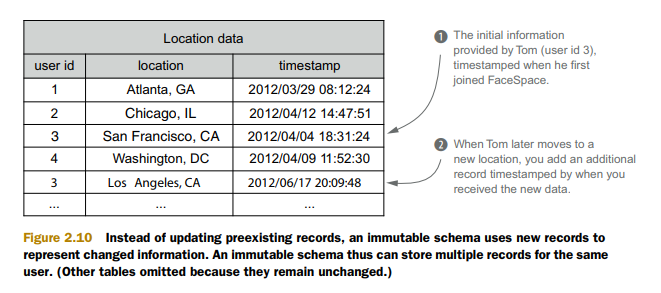
Tom debe mudarse a Los Ángeles, actualizar la entrada resaltada para reflejar su ubicación actual, pero en el proceso, también perderías todo el conocimiento que Tom ha vivido en San Francisco.

Con un esquema inmutable, las cosas se ven diferentes. En lugar de almacenar una instantánea actual del mundo, como hace el esquema mutable, se crea un registro separado cada vez que evoluciona la información del usuario. Lograr esto requiere dos cambios. En primer lugar, que registra cada campo de información del usuario en una tabla separada. En segundo lugar, hacer cada unidad de datos a un momento cuando la información se sabe para ser verdad. Figura 2.9 se muestra un esquema inmutable correspondiente para almacenar información de FaceSpace.

Tom primero se unió a FaceSpace el 04 de abril de 2012 y su información de perfil. El tiempo que aprende estos datos se refleja en la fecha y hora del registro. Cuando posteriormente se traslada a Los Ángeles en 17 de junio de 2012, añadir un nuevo registro a la tabla de almacén, sellos por cuando él cambió su perfil — véase la figura 2.10.

Ahora tiene dos registros de ubicación de Tom (usuario ID #3), y porque las unidades de datos están ligadas a momentos particulares, que pueden ser verdadero. Ubicación actual de Tom consiste en una simple consulta de datos: buscar en todos los lugares y elegir el uno con el de fecha más reciente. Manteniendo cada campo en una tabla separada, sólo se registre la información que cambió. Esto requiere menos espacio de almacenamiento y garantías que cada registro es nueva información y no simplemente prorrogados desde el último registro.

Una de las ventajas y desventajas del enfoque inmutable es que utiliza almacenamiento de información más que un esquema mutable. En primer lugar, el ID de usuario se especifica para cada propiedad, en lugar de sólo una vez por fila, como con un enfoque mutable. Además, se almacena toda la historia de los acontecimientos en lugar de sólo la visión actual del mundo. Pero no por nada Big Data llaman "Big Data". Usted debe tomar ventaja de la capacidad de almacenar grandes cantidades de datos utilizando tecnologías de Big Data para obtener los beneficios de la inmutabilidad. No se puede exagerar la importancia de tener un simple y fuertemente humano falla tolerante principal conjunto de datos.



2.1.3 Data is eternally true

La consecuencia clave de inmutabilidad es que cada dato es verdadero a perpetuidad. Es decir, un pedazo de datos, una vez que es verdad, debe ser siempre verdadero. Inmutabilidad no tendría sentido sin esta propiedad, y vio cómo marcado cada pedazo de datos con una marca de tiempo es una forma práctica para hacer datos eternamente verdadero.

Esta mentalidad es la misma que cuando has aprendido historia en la escuela. El hecho de los Estados Unidos consistió en trece Estados en 04 de julio de 1776, siempre es así debido a la fecha específica; el hecho de que el número de Estados ha aumentado desde entonces se captura en datos adicionales (también perpetuos).

En general, el conjunto de datos principal crece constantemente añadiendo nuevas piezas inmutables y eternamente verdaderas de datos. Hay algunos casos especiales, sin embargo, en la que borrar datos, y estos casos no son incompatibles con datos siendo eternamente verdad. Vamos a considerar los casos:

* Recolección de basura, cuando se realiza una recolección de basura, eliminar todas las unidades de datos que tienen escaso valor. Puede utilizar la recolección de basura para implementar políticas de dataretention que controlan el crecimiento del conjunto de datos principal. Por ejemplo, usted puede decidir implementar una política que mantiene sólo una ubicación por persona por año en vez la historia completa de cada vez que un usuario cambia localizaciones.
* Regulaciones: las regulaciones gubernamentales pueden requerir que purgar datos de tus bases de datos bajo ciertas condiciones.

En ambos casos, eliminar los datos no es una declaración acerca de la veracidad de los datos. En cambio, es una declaración sobre el valor de los datos. Aunque los datos están eternamente verdaderos, usted puede preferir a "olvidar" la información porque debe o porque no proporciona el suficiente valor para el costo de almacenamiento de información.

Procederemos introduciendo un modelo de datos que utiliza estas propiedades claves de los datos.

¿Eliminación de datos inmutables? Usted se estará preguntando cómo es posible eliminar datos inmutables. Aparentemente, esto parece una contradicción. Es importante distinguir que el eliminar a que nos referimos es un caso especial y raro. En uso normal, datos están inmutables, y aplicar esa propiedad por acciones tales como establecer los permisos adecuados. Eliminación de datos es rara, puede tomarse el máximo cuidado para asegurar que se haga con seguridad. Creemos que eliminar datos más segura se logra mediante la producción de una segunda copia de la base de datos principal con los datos que filtra, ejecutando trabajos analíticos para verificar que los datos correctos se filtraron y entonces y sólo entonces reemplazar la antigua versión de la maestría conjunto de datos.

2.2 the fact-based model for representing data

Datos están el conjunto de información que no puede derivar de cualquier otra cosa, pero hay muchas maneras que usted podría elegir representar dentro del conjunto de datos principal. Además de tradicionales tablas relacionales, XML estructurada y semiestructurados documentos JSON son otras posibilidades para almacenar datos. Sin embargo, recomendamos el modelo hecho para este propósito. En el modelo basado en los hechos, deconstruyen los datos en unidades fundamentales llamadas (como era de esperar) hechos.

En la discusión de inmutabilidad, tienes un vistazo del modelo basado en los hechos, en que el principal conjunto de datos crece continuamente con la incorporación de datos inmutables, de sellos. Ahora podrá ampliar sobre lo que ya discutido para explicar el modelo hecho en su totalidad. Nosotros primero introducir el modelo en el contexto del ejemplo FaceSpace y discutir sus propiedades básicas. Luego continuaremos con discutir cómo y por qué usted debe hacer sus datos identificables. Para terminar, vamos a explicar los beneficios de usar el modelo de hecho y por qué es un excelente elección para su conjunto de datos.

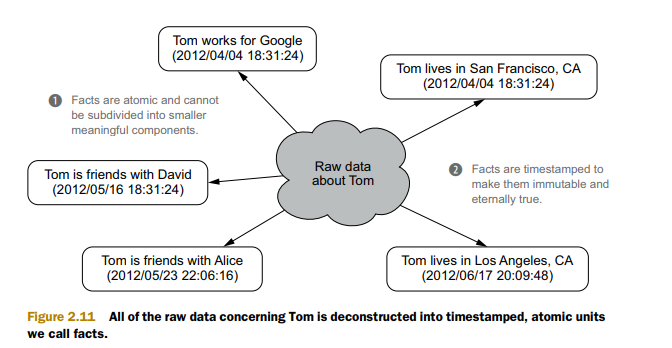
2.2.1 Example facts and their propetiers

Figura 2.11 muestra ejemplos de hechos acerca de Tom de los datos de FaceSpace, así como dos propiedades de datos de la base — son atómicos y sellos.

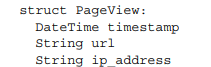
Hechos son atómicas porque no puede ser subdivididos más lejos en componentes significativos. Datos colectivos, tales como lista de amigo de Tom en la figura, se representan como varios hechos independientes. Como consecuencia de ser atómico, hay no hay redundancia de la información a través de distintos hechos. Hechos con las marcas de tiempo deben venir como ninguna sorpresa, dada nuestra discusión anterior sobre los datos, las marcas de hora hacen cada hecho inmutable y eternamente verdadero.

Estas propiedades hacen del modelo basado en los hechos un modelo sencillo y expresivo para el conjunto de datos, sin embargo, hay una propiedad adicional se recomienda imponer en sus hechos: identificabilidad.

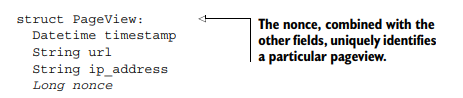
CÓMO HACER IDENTIFICACIÓN DE LOS HECHOS Además de ser atómicos y con marca de tiempo, los datos deben asociarse con una parte de datos identificable de manera única. Esto se explica más fácilmente con el ejemplo.



Supongamos que desea almacenar datos sobre vistas de página en FaceSpace. Tu primer acercamiento Podría verse algo como esto (en pseudo-código):



Los hechos que utilizan esta estructura no identifican de forma única un evento de visita de página en particular. Si es múltiple las visitas de página entran al mismo tiempo para la misma URL desde la misma dirección IP, Cada página vista tendrá exactamente el mismo registro de datos. En consecuencia, si te encuentras con dos registros de páginas vistas idénticos, no hay manera de saber si se refieren a dos eventos o si se introdujo accidentalmente una entrada duplicada en su conjunto de datos.  Para distinguir diferentes vistas de página, puede agregar un nonce a su esquema: un bit de 64 bits. Número generado aleatoriamente para cada página vista:



La adición del nonce permite distinguir eventos de vista de página de cada otra, y si dos unidades de datos de vista de página son idénticas (todos los campos, incluido el nonce), Usted sabe que se refieren exactamente al mismo evento.

Hacer hechos identificables significa que puede escribir el mismo hecho en el conjunto de datos maestro Varias veces sin cambiar la semántica del conjunto de datos maestro. Tus consultas Puede filtrar los hechos duplicados al hacer sus cálculos. Como resulta, y como verá más adelante, tener hechos distinguibles hace que la implementación del resto de Lambda Architecture mucho más fácil.

Los duplicados no son tan raros como podrías pensar

A primera vista, puede que no sea obvio por qué nos importa tanto la identidad y los duplicados. Después de todo, para evitar duplicados, la primera inclinación sería asegurar que un El evento se registra una sola vez. Desafortunadamente, la vida no siempre es tan simple cuando se trata con Big Data.

Una vez que FaceSpace se convierta en un éxito, requerirá cientos, y luego miles, de servidores web. La construcción del conjunto de datos maestro requerirá la agregación de los datos de cada uno de estos Servidores a un sistema central, ninguna tarea trivial. Existen herramientas de recolección de datos adecuadas. para esta situación (Scribe, Apache Flume, syslog-ng de Facebook y muchos otros) pero cualquier solución debe ser tolerante a fallas.

Un "fallo" común que estos sistemas deben anticipar es una partición de red donde el el almacén de datos de destino deja de estar disponible. Para estas situaciones, sistemas tolerantes a fallos. comúnmente manejan las operaciones fallidas reintentando hasta que tengan éxito. Porque el el remitente no sabrá qué datos se recibieron por última vez, un enfoque estándar es reenviar Todos los datos aún no han sido reconocidos por el destinatario. Pero si parte del intento original lo hizo Llegue a Metastore, terminará con duplicados en su conjunto de datos.

Hay formas de hacer transaccionales este tipo de operaciones, pero puede ser bastante Difícil y conllevan costes de rendimiento. Una parte importante para asegurar la corrección en su Los sistemas están evitando soluciones difíciles. Al abrazar hechos distinguibles, eliminas la necesidad de anexos transaccionales al conjunto de datos maestro y que sea más fácil de razonar Sobre la corrección del sistema completo. Después de todo, ¿por qué colocar cargas difíciles sobre ti mismo? ¿Cuando un pequeño ajuste a su modelo de datos puede evitar esos desafíos por completo?

Para resumir rápidamente, el modelo basado en hechos

■ Almacena sus datos en bruto como hechos atómicos

■ Mantiene los hechos inmutables y eternamente verdaderos mediante el uso de marcas de tiempo

■ Asegura que cada hecho sea identificable para que el procesamiento de consultas pueda identificar duplicados

A continuación, analizaremos los beneficios de elegir el modelo basado en hechos para su conjunto de datos maestro.

2.2.2 Benefits of the fact-based model

Con un modelo basado en hechos, el conjunto de datos maestro será una lista cada vez mayor de inmutables, Hechos atómicos. Este no es un patrón para el que se crearon bases de datos relacionales, si provienen de un fondo relacional, su cabeza puede estar girando. La buena noticia es que al cambiar el paradigma de tu modelo de datos ganas numerosas ventajas. Específicamente, tu información

■ Es consultable en cualquier momento de su historia.

■ Tolera errores humanos.

■ Maneja información parcial.

■ Tiene las ventajas de las formas normalizadas y desnormalizadas.

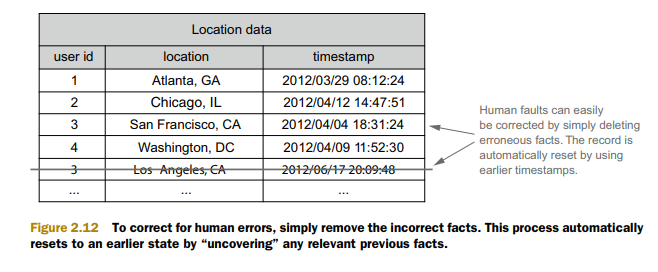
Veamos cada una de estas ventajas a su vez.

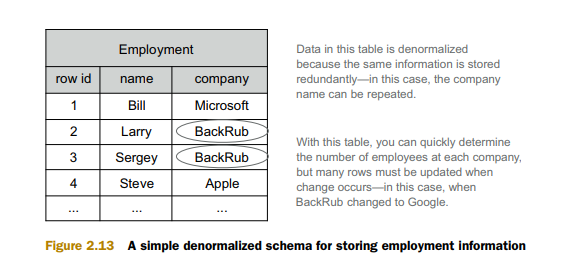
EL DATASET ES CONSIDERABLE EN CUALQUIER MOMENTO EN SU HISTORIA

En lugar de almacenar solo el estado actual del mundo, como lo haría con un mutable, esquema relacional, tiene la capacidad de consultar sus datos en cualquier momento cubierto por su conjunto de datos Esta es una consecuencia directa de que los hechos están marcados por el tiempo y son inmutables. Las "actualizaciones" y las "eliminaciones" se realizan agregando nuevos datos con marcas de tiempo más recientes, pero como en realidad no se eliminan datos, puede reconstruir el estado de la mundo a la hora especificada por su consulta.

LOS DATOS ES TOLERANTE HUMANO-FALLO La tolerancia a las fallas humanas se logra simplemente eliminando cualquier hecho erróneo. Supongamos que usted guardó erróneamente que Tom se mudó de San Francisco a Los Ángeles; vea la figura 2.12.  Al eliminar el hecho de Los Ángeles, la ubicación de Tom se "reinicia" automáticamente porque El hecho de San Francisco se convierte en la información más reciente. LA BASE DE DATOS MANEJA FÁCILMENTE LA INFORMACIÓN PARCIAL El almacenamiento de un hecho por reg0istro facilita el manejo de información parcial sobre una entidad sin introducir valores NULL en su conjunto de datos. Supongamos que Tom proporcionó su edad Y el género pero no su ubicación o profesión. Tu conjunto de datos solo tendría datos para la información conocida: cualquier hecho "ausente" sería lógicamente equivalente a NULL. La información adicional que Tom proporciona en un momento posterior se presentará naturalmente A través de nuevos hechos. EL ALMACENAMIENTO DE DATOS Y LAS CAPAS DE PROCESAMIENTO DE LA CONSULTA SON INDEPENDIENTES Hay otra ventaja clave del modelo basado en hechos que se debe en parte a la estructura de la propia arquitectura lambda. Almacenando la información tanto en el lote. y las capas de servicio, tiene la ventaja de mantener sus datos tanto en condiciones normalizadas como Formas desnormalizadas y cosechando los beneficios de ambos.

LA NORMALIZACIÓN ES UN TÉRMINO SOBRECARGADO La normalización de los datos no tiene ninguna relación con el término de normalización semántica que usamos anteriormente. En este caso, la normalización de datos se refiere al almacenamiento de datos de manera estructurada para minimizar la redundancia y promover la coherencia.

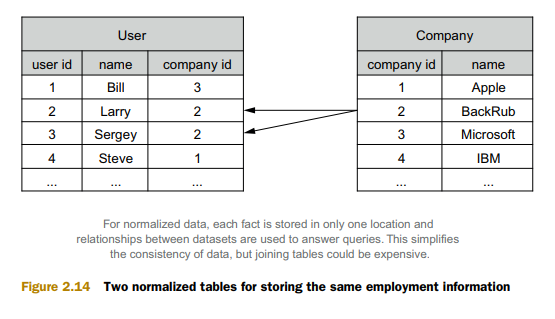




Pongamos el escenario con un ejemplo que incluya tablas relacionales: el contexto donde los datos La normalización es la más frecuente. Las tablas relacionales requieren que elijas entre esquemas normalizados y desnormalizados basados ​​en lo que es más importante para Usted: la eficiencia de la consulta o la consistencia de los datos. Supongamos que querías guardar el empleo. Información para varias personas de interés. La figura 2.13 ofrece un simple desnormalizado. esquema adecuado para este fin.

En este esquema desnormalizado, el mismo nombre de la compañía podría potencialmente almacenarse en múltiples filas. Esto le permitiría determinar rápidamente el número de empleados para cada compañía, pero necesitaría actualizar muchas filas en caso de que una compañía cambia su nombre Tener la información almacenada en múltiples ubicaciones aumenta el riesgo de se vuelve inconsistente.

 En comparación, considere el esquema normalizado en la figura 2.14.  Los datos en un esquema normalizado se almacenan en una sola ubicación. Si BackRub debería cambie su nombre a Google, hay una sola fila en la tabla de la Compañía que debe ser alterado Esto elimina el riesgo de inconsistencia, pero debe unirse a las tablas para responder Consultas: un cálculo potencialmente costoso.



La elección mutuamente exclusiva entre esquemas normalizados y desnormalizados es necesaria porque, para las bases de datos relacionales, las consultas se realizan directamente en los datos en El nivel de almacenamiento. Por lo tanto, debe sopesar la importancia de la eficiencia de la consulta frente a consistencia de los datos y elegir entre los dos tipos de esquema.

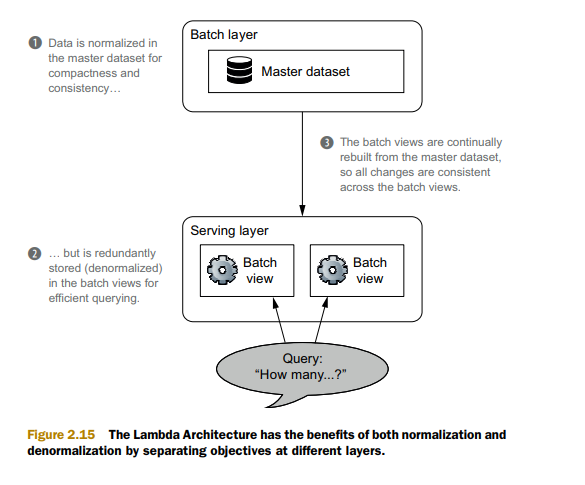
 En contraste, los objetivos del procesamiento de consultas y el almacenamiento de datos están claramente separados en la arquitectura lambda. Eche un vistazo a las capas de lotes y servidores en la figura 2.15.

 En la arquitectura Lambda, el conjunto de datos maestro está completamente normalizado. Como viste en el discusión del modelo basado en hechos, no se almacenan datos de forma redundante. Las actualizaciones son fáciles manejado porque agregar un hecho nuevo con una marca de tiempo actual "reemplaza" a cualquier anterior Hechos relacionados.

De manera similar, las vistas de lotes son como tablas desnormalizadas en ese único dato de El conjunto de datos maestro puede ser indexado en muchas vistas de lote. La diferencia clave es que la

Las vistas por lotes se definen como funciones en el conjunto de datos maestro. En consecuencia, no hay necesidad para actualizar una vista por lotes porque se reconstruirá continuamente a partir del conjunto de datos maestro. Esta tiene el beneficio adicional de que las vistas por lotes y el conjunto de datos maestro nunca estarán fuera de sincronizar La arquitectura Lambda le brinda los beneficios conceptuales de la normalización completa. con los beneficios de rendimiento de la indexación de datos de diferentes maneras para optimizar las consultas.

 En resumen, todos estos beneficios hacen del modelo basado en hechos una excelente elección. para su conjunto de datos maestro. Pero eso es suficiente discusión a nivel teórico; sumergirse en los detalles de la implementación práctica de un modelo de datos basado en hechos.



2.3 Graph schemas

Cada hecho dentro de un modelo basado en hechos captura una sola pieza de información. Pero el los hechos por sí solos no transmiten la estructura detrás de los datos. Es decir, no hay descripción. de los tipos de hechos contenidos en el conjunto de datos, ni ninguna explicación de las relaciones entre ellos. En esta sección, presentaremos esquemas gráficos, gráficos que capturan los Estructura de un conjunto de datos almacenado utilizando el modelo basado en hechos. Discutiremos los elementos de un esquema gráfico y la necesidad de hacer un esquema ejecutable.

Comencemos por estructurar primero nuestros hechos de FaceSpace como un gráfico.

2.3.1 Elementos of a graph schema

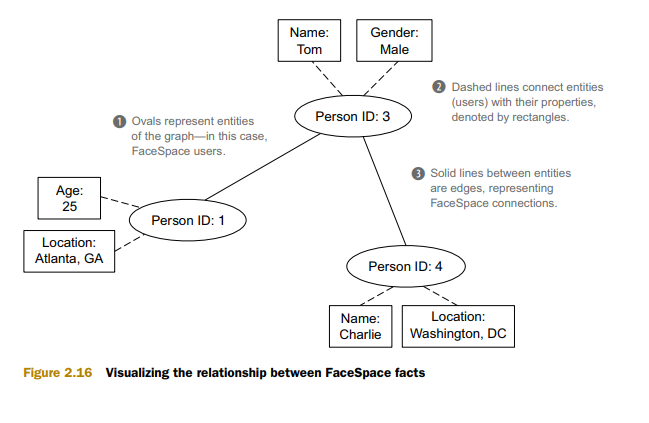
En la última sección discutimos los hechos de FaceSpace con gran detalle. Cada hecho representa ya sea una información sobre un usuario o una relación entre dos usuarios. Figura 2.16 representa un esquema gráfico que representa las relaciones entre el FaceSpace hechos. Proporciona una visualización útil de sus usuarios, sus usuarios individuales. La información, y las amistades entre ellos.

La figura resalta los tres componentes centrales de un esquema gráfico: nodos, bordes, y propiedades:

■ Los nodos son las entidades en el sistema. En este ejemplo, los nodos son el FaceSpace. usuarios, representados por una identificación de usuario. Como otro ejemplo, si FaceSpace permite a los usuarios para identificarse como parte de un grupo, entonces los grupos también estarían representados por nodos.

■ Los bordes son relaciones entre nodos. La connotación en FaceSpace es directa, una El borde entre usuarios representa una amistad de FaceSpace. Más tarde, podría agregar tipos de bordes adicionales entre los usuarios para identificar compañeros de trabajo, familia miembros, o compañeros de clase.

■ Las propiedades son información sobre entidades. En este ejemplo, edad, género, ubicación y Todas las demás piezas de información individual son propiedades.



LOS BORDES ESTÁN ESTRICTAMENTE ENTRE LOS NODOS A pesar de que las propiedades y los nodos son Conectadas visualmente en la figura, estas líneas no son aristas. Estan presentes sólo para ayudar a ilustrar la asociación entre los usuarios y su información personal. Denotamos la diferencia mediante el uso de líneas continuas para bordes y discontinuas. Líneas para conexiones de propiedad.

El esquema gráfico proporciona una descripción completa de todos los datos contenidos en un conjunto de datos A continuación, discutiremos la necesidad de garantizar que todos los hechos dentro de un conjunto de datos se ajusten rígidamente.se adhieren al esquema.

2.3.2 The need for an enforceable schema

En este punto, la información se almacena como hechos y un esquema gráfico describe los tipos de Hechos contenidos en el conjunto de datos. Estás todo listo, ¿verdad? Bueno, no del todo. Usted todavía necesita decide en qué formato almacenarás tus datos.

 Una primera idea podría ser utilizar un formato de texto semiestructurado como JSON. Esto proporcionaría simplicidad y flexibilidad, permitiendo esencialmente que cualquier cosa sea escrita al maestro conjunto de datos Pero en este caso es demasiado flexible para nuestras necesidades.

 Para ilustrar este problema, suponga que eligió representar la edad de Tom utilizando JSON:



No hay problemas con la representación de este hecho único, pero no hay manera de Asegúrese de que todos los hechos posteriores seguirán el mismo formato. Como resultado de humanos error, el conjunto de datos también podría incluir hechos como estos:



Ambos ejemplos son JSON válidos, pero tienen formatos inconsistentes o faltan datos. En particular, en la sección anterior destacamos la importancia de tener una marca de tiempo para cada hecho, pero un formato de texto no puede hacer cumplir este requisito. Para efectivamente Utilice sus datos, debe proporcionar garantías sobre el contenido de su conjunto de datos.

La alternativa es usar un esquema ejecutable que defina rigurosamente la estructura de tus hechos. Los esquemas ejecutables requieren un poco más de trabajo por adelantado, pero Garantizar que todos los campos requeridos estén presentes y garantizar que todos los valores sean del esperado.tipo. Con estas garantías, un desarrollador estará seguro de qué datos pueden esperar: que cada hecho tendrá una marca de tiempo, que el nombre de un usuario siempre será una cadena, Etcétera.

La clave es que cuando se comete un error al crear un dato, un el esquema ejecutable dará errores en ese momento, en lugar de cuando alguien intente usar los datos más tarde en un sistema diferente. Cuanto más cerca del error aparece el error, el Más fácil es atrapar y arreglar.

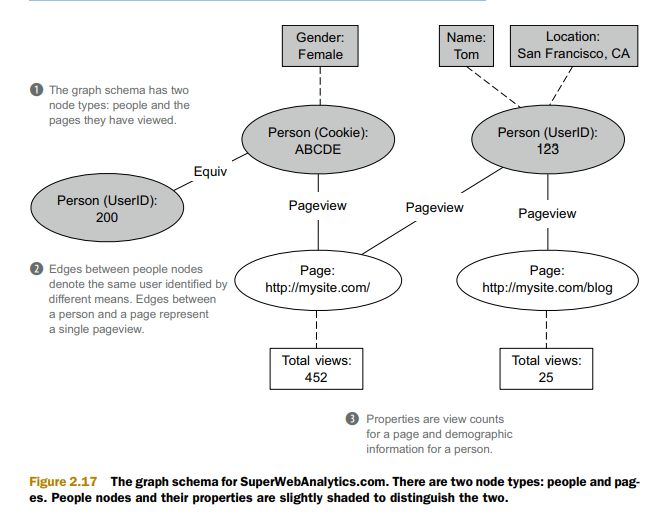
 En el siguiente capítulo, verá cómo implementar un esquema ejecutable utilizando un marco de serialización. Un marco de serialización proporciona una manera neutral de lenguaje para Defina los nodos, bordes y propiedades de su esquema. Luego genera código (potencialmente en muchos idiomas diferentes) que serializa y deserializa los objetos en su esquema para que puedan almacenarse y recuperarse de su conjunto de datos maestro.

 Somos conscientes de que en este momento es posible que tenga hambre de detalles. No se preocupe, nosotros Creer que la mejor manera de aprender es haciendo. En la siguiente sección diseñaremos los hechos modelo para SuperWebAnalytics.com en su totalidad, y en el siguiente capítulo veremos implementarlo utilizando un marco de serialización.

2.4 a complete data model for superwebanalytics.com

En esta sección, nuestro objetivo es unir todo el material del capítulo utilizando el Ejemplo de SuperWebAnalytics.com. Comenzaremos con la figura 2.17, que contiene una Esquema gráfico adecuado para nuestro propósito.

 En este esquema hay dos tipos de nodos: personas y páginas. Como puedes ver, hay Hay dos categorías distintas de nodos de personas para distinguir personas con una identidad conocida. de personas con las que solo puede identificarse mediante una cookie de navegador web.



Los bordes en el esquema son bastante simples. Un borde de página se produce entre una persona y una página para cada vista distinta, mientras que se produce un borde equivalente entre dos nodos de persona Cuando representan al mismo individuo. Esto último ocurriría cuando una persona inicialmente identificado por solo una cookie se identifica completamente en un momento posterior.

 Las propiedades también se explican por sí mismas. Las páginas tienen un total de visitas a la página, y las personas Tener información demográfica básica: nombre, género y ubicación.

 Una de las bellezas de los esquemas de modelos y gráficos basados ​​en hechos es que pueden Evolucionar a medida que diferentes tipos de datos estén disponibles. Un esquema gráfico proporciona una interfaz consistente con datos arbitrariamente diversos, por lo que es fácil incorporar nuevos tipos de información. Las adiciones de esquema se realizan mediante la definición de un nuevo nodo, borde y propiedad tipos Debido a la atomicidad de los hechos, estas adiciones no afectan a las ya existentes.tipos de hechos

2.5 Summary

Cómo modelar su conjunto de datos maestro crea la base para su sistema Big Data. Las decisiones tomadas sobre el conjunto de datos maestro determinan el tipo de análisis que puede realizar en sus datos y cómo va a consumir esos datos. La estructura del maestro. El conjunto de datos debe admitir la evolución de los tipos de datos almacenados, debido a que Los tipos de datos pueden cambiar considerablemente con los años.

 El modelo basado en hechos proporciona una representación simple pero expresiva de sus datos naturalmente manteniendo una historia completa de cada entidad en el tiempo. Su naturaleza sólo apéndice. facilita su implementación en un sistema distribuido y puede evolucionar fácilmente a medida que sus datos y tus necesidades cambian. No solo estás implementando un sistema relacional de una manera más de forma escalable: también está agregando capacidades completamente nuevas a su sistema.

3. DATA MODEL FOR BIG DATA: ILLUSTRACION

En el último capítulo vio los principios de la formación de un modelo de datos: el valor de datos en bruto, que se ocupan de la normalización semántica, y la importancia crítica de inmutabilidad. Usted vio cómo un esquema gráfico puede satisfacer todas estas propiedades y vio cómo se ve el esquema gráfico para SuperWebAnalytics.com.

Este es el primero de los capítulos de ilustración, en el que demostramos los conceptos de El capítulo anterior utilizando herramientas del mundo real. Puedes leer solo los capítulos teóricos de El libro y aprender toda la arquitectura Lambda, pero los capítulos de ilustración muestran Te los matices de mapear la teoría al código real. En este capítulo implementaremos el modelo de datos de SuperWebAnalytics.com que utiliza Apache Thrift, un marco de serialización. Verá que incluso en una tarea tan sencilla como escribir un esquema, hay fricción Entre la teoría idealizada y lo que se puede lograr en la práctica.

3.1 ¿Por qué un marco de serialización?

Muchos desarrolladores siguen el camino de escribir sus datos en bruto en un formato sin esquemas como JSON. Esto es atractivo debido a lo fácil que es comenzar, pero este enfoque rápidamente conduce a los problemas. Ya sea debido a errores o malentendidos entre diferentes desarrolladores, la corrupción de datos inevitablemente se produce. Es nuestra experiencia que la corrupción de datos Los errores son algunos de los que requieren más tiempo para depurar.

Los problemas de corrupción de datos son difíciles de depurar porque tiene muy poco contexto en Cómo ocurrió la corrupción. Normalmente solo notará que hay un problema cuando hay un error downstream en el procesamiento, mucho después de que se escribieron los datos corruptos. Por ejemplo, puede obtener una excepción de puntero nulo debido a un campo obligatorio estar desaparecido Te darás cuenta rápidamente de que el problema es un campo faltante, pero tendrás No hay absolutamente ninguna información sobre cómo llegaron esos datos allí en primer lugar.

Cuando crea un esquema ejecutable, obtiene errores al momento de escribir el

Datos: le proporciona un contexto completo sobre cómo y por qué los datos se volvieron inválidos (como una pila rastro). Además, el error evita que el programa dañe el conjunto de datos maestro escribiendo esos datos.

Los marcos de serialización son un enfoque fácil para hacer un esquema ejecutable.

Si alguna vez ha utilizado un lenguaje de tipo estático orientado a objetos, utilizando una serialización El marco será inmediatamente familiar. Marcos de serialización generan código para cualquier idioma que desee utilizar para leer, escribir y validar objetos que coincide con su esquema.

Sin embargo, los marcos de serialización son limitados cuando se trata de lograr una

esquema riguroso. Después de discutir cómo aplicar un marco de serialización a SuperWebAnalytics.com modelo de datos, discutiremos estas limitaciones y cómo solucionar ellos.

3.2 Apache Thrift

Apache Thrift (http://thrift.apache.org/) es una herramienta que se puede usar para definir estáticamente esquemas mecanografiados, ejecutables. Proporciona un lenguaje de definición de interfaz para describir la esquema en términos de tipos de datos genéricos, y esta descripción se puede usar posteriormente para Generar la implementación real en múltiples lenguajes de programación.

NUESTRO USO DE APACHE THRIFT Thrift se desarrolló inicialmente en Facebook para la construcción de servicios en varios idiomas. Se puede usar para muchos propósitos, pero Limitar nuestra discusión a su uso como un marco de serialización.

Otros marcos de serialización Hay otras herramientas similares a Apache Thrift, como Protocol Buffers y Avro. Recuerde, el propósito de este libro no es proporcionar una encuesta de todas las herramientas posibles. para cada situación, pero para utilizar una herramienta adecuada para ilustrar los conceptos fundamentales. Como marco de serialización, Thrift es práctico, probado exhaustivamente y ampliamente utilizado

Los caballos de batalla de Thrift son las definiciones de estructura y tipo de unión. Están compuestas de otros campos, tales como

■ Tipos de datos primitivos (cadenas, enteros, largos y dobles)

■ Colecciones de otros tipos (listas, mapas y conjuntos)

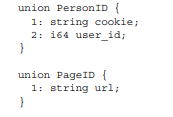
■ Otras estructuras y uniones

En general, las uniones son útiles para representar nodos, las estructuras son representaciones naturales De bordes, y propiedades utilizan una combinación de ambos. Esto se hará evidente. de las definiciones de tipo necesarias para representar el esquema de SuperWebAnalytics.com componentes.

3.2.1 Nodes

Para nuestros nodos de usuario SuperWebAnalytics.com, una persona se identifica ya sea por un ID de usuario o una cookie del navegador, pero no ambas. Este patrón es común para los nodos, y coincide exactamente con un tipo de datos de unión: un solo valor que puede tener cualquiera de varios representaciones.

En Thrift, los sindicatos se definen enumerando todas las representaciones posibles. El seguimiento El código define los nodos de SuperWebAnalytics.com mediante uniones de ahorro:



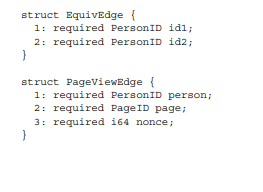
Tenga en cuenta que las uniones también se pueden utilizar para nodos con una sola representación. Uniones Permitir que el esquema evolucione a medida que evolucionan los datos. Discutiremos esto más adelante en este sección.

3.2.2 Edges

Cada borde se puede representar como una estructura que contiene dos nodos. El nombre de una ventaja struct indica la relación que representa, y los campos en la estructura de borde contienen

Las entidades involucradas en la relación.

 La definición del esquema es muy simple:

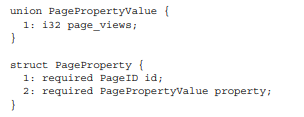


Los campos de una estructura Thrift se pueden denotar como requeridos u opcionales. Si un campo es definido como requerido, entonces se debe proporcionar un valor para ese campo, o si no, Thrift Dar un error en la serialización o deserialización. Debido a que cada borde en un gráfico El esquema debe tener dos nodos, son campos obligatorios en este ejemplo.

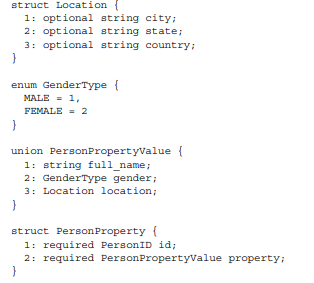
3.2.3 Properties

Por último, vamos a definir las propiedades. Una propiedad contiene un nodo y un valor para la propiedad. El valor puede ser uno de muchos tipos, por lo que se representa mejor utilizando una estructura de unión.

Empecemos definiendo el esquema para las propiedades de la página. Solo hay una propiedad para las páginas, así que es muy simple:



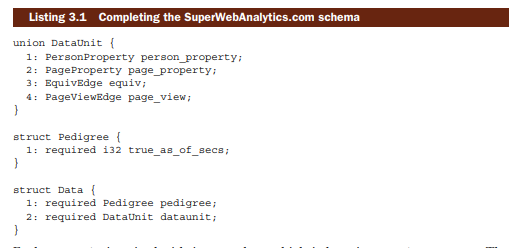
A continuación vamos a definir las propiedades para las personas. Como se puede ver, la propiedad de ubicación es Más complejo y requiere que se defina otra estructura:



La estructura de ubicación es interesante porque los campos ciudad, estado y país podrían tener ha sido almacenado como datos separados. En este caso, están tan estrechamente relacionados que hace sentido de ponerlos todos en una estructura como campos opcionales. Al consumir ubicación información, casi siempre querrá todos esos campos.

3.2.4 Atando todo en objetos de datos

En este punto, los bordes y las propiedades se definen como tipos separados. Lo ideal sería que desea almacenar todos los datos juntos para proporcionar una interfaz única para acceder a su información. Además, también hace que sus datos sean más fáciles de administrar si se almacenan en una sola conjunto de datos Esto se logra envolviendo cada propiedad y tipo de borde en un DataUnit union: consulte la siguiente lista de códigos.



Cada unidad de datos está emparejada con sus metadatos, que se guardan en una estructura de pedigrí. Los pedigree contiene la marca de tiempo para la información, pero también podría contener Información de depuración o la fuente de los datos. La estructura de datos final corresponde a un hecho del modelo basado en hechos.

3.2.5 Envolving for schema

Thrift está diseñado para que los esquemas puedan evolucionar con el tiempo. Esta es una propiedad crucial, porque a medida que cambien los requisitos de su negocio, deberá agregar nuevos tipos de datos,

y querrás hacerlo tan fácilmente como sea posible.

La clave para la evolución de los esquemas de Thrift son los identificadores numéricos asociados con cada uno campo. Esos ID se utilizan para identificar campos en su forma serializada. Cuando quieras

cambia el esquema pero aún así es compatible hacia atrás con los datos existentes, debes obedecer

las siguientes reglas:

■ Los campos pueden ser renombrados. Esto se debe a que la forma serializada de un objeto usa el

ID de campo, no los nombres, para identificar campos.

■ Es posible que se elimine un campo, pero nunca debe volver a usar esa ID de campo. Cuando se deserializa datos existentes, Thrift ignorará todos los campos con ID de campo no incluidos en el

esquema. Si tuviera que reutilizar una ID de campo previamente eliminada, Thrift intentaría

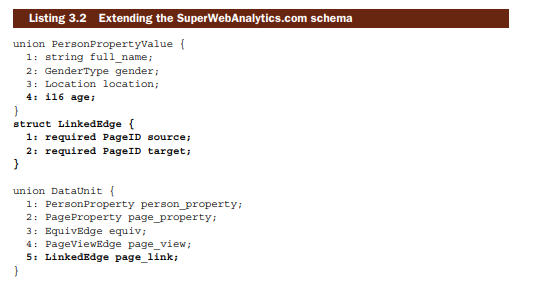
Deserializar los datos antiguos en el nuevo campo, lo que llevará a que sea inválido o Datos Incorrectos.

■ Solo se pueden agregar campos opcionales a las estructuras existentes. No puedes agregar campos obligatorios porque los datos existentes no tendrán esos campos y, por lo tanto, no serán deserializables.

(Tenga en cuenta que esto no se aplica a los sindicatos, porque los sindicatos no tienen noción de

campos obligatorios y opcionales.)

Como ejemplo, si desea cambiar el esquema de SuperWebAnalytics.com a almacenar la edad de una persona y los enlaces entre las páginas web, usted haría lo siguiente cambios en su archivo de definición de Thrift (cambios en negrita).



Tenga en cuenta que la adición de una propiedad new age se realiza agregándola a la correspondiente

estructura de unión, y se incorpora una nueva ventaja agregándola a la unión DataUnit.

3.3 Limitaciones de los marcos de serialización.

Los marcos de serialización solo comprueban que todos los campos requeridos están presentes y son de la Tipo esperado. No pueden verificar propiedades más ricas como "Las edades no deben ser negativas" o “las marcas de tiempo reales no deben estar en el futuro”. Los datos no coinciden estas propiedades indicarían un problema en su sistema, y ​​no querría Escrito a su conjunto de datos maestro.

Esto puede no parecer una limitación porque los marcos de serialización parecen algo similar a cómo funcionan los esquemas en bases de datos relacionales. De hecho, puede haber encontrado Los esquemas de bases de datos relacionales son un dolor para trabajar y se preocupan de que hacer esquemas incluso Más estricto sería aún más doloroso. Pero te instamos a que no confundas lo incidental. complejidades de trabajar con esquemas de bases de datos relacionales con el valor de los esquemas sí mismos. Las dificultades para representar objetos anidados y hacer migraciones de esquemas. con bases de datos relacionales no existen al aplicar marcos de serialización Representar objetos inmutables mediante esquemas gráficos.

La forma correcta de pensar acerca de un esquema es como una función que toma datos.

y devuelve si es válido o no. El lenguaje de esquema para Apache Thrift te permite representan un subconjunto de estas funciones donde solo existen campos y tipos de campos comprobado. La herramienta ideal le permitiría implementar cualquier función de esquema posible.

Tal herramienta ideal, particularmente una que es neutral al lenguaje, no existe, pero

Hay dos enfoques que puede tomar para solucionar estas limitaciones con una serialización.

Marco como Apache Thrift:

■ Envuelva su código generado en un código adicional que verifique las propiedades adicionales que le interesan sobre, como las edades no son negativas. Este enfoque funciona bien siempre y cuando solo estés leer / escribir datos de / a un solo idioma, si utiliza varios idiomas, Tienes que duplicar la lógica en muchos idiomas.

■ Compruebe las propiedades adicionales al comienzo de su flujo de trabajo de procesamiento por lotes. Esta El paso dividiría su conjunto de datos en "datos válidos" y "datos no válidos" y enviaría una notificación si se encontró algún dato no válido. Este enfoque hace que sea más fácil de implementar El resto de su flujo de trabajo, ya que cualquier cosa pasa la verificación de validez. Se puede asumir que tiene las propiedades más estrictas que le interesan. Pero este enfoque no impide que los datos no válidos se escriban en el conjunto de datos maestro y no ayuda a determinar el contexto en el que ocurrió la corrupción.

Ninguno de los dos enfoques es ideal, pero es difícil ver cómo puede hacerlo mejor si su organización

Lee / escribe datos en múltiples idiomas. Tienes que decidir si prefieres Mantenga la misma lógica en varios idiomas o pierda el contexto en el que se produce la corrupción. Fue presentado. El único enfoque que sería perfecto sería una serialización. marco que también es un lenguaje de programación de propósito general que se traduce en los idiomas a los que se dirige. Tal herramienta no existe, aunque teóricamente es posible.

3.4 Summary

En su mayor parte, la implementación del esquema gráfico ejecutable para SuperWebAnalytics.com

era sencillo Viste la fricción que aparece al usar una serialización marco para este propósito, es decir, la incapacidad de hacer cumplir todas las propiedades que preocuparse. Las herramientas rara vez capturarán sus requisitos perfectamente, pero es importante Para saber qué sería posible con las herramientas ideales. De esa manera usted es consciente de la está haciendo concesiones y puede estar pendiente de mejores herramientas (o hacer las suyas propias). Este será un tema común a medida que pasemos por los capítulos de teoría e ilustración.

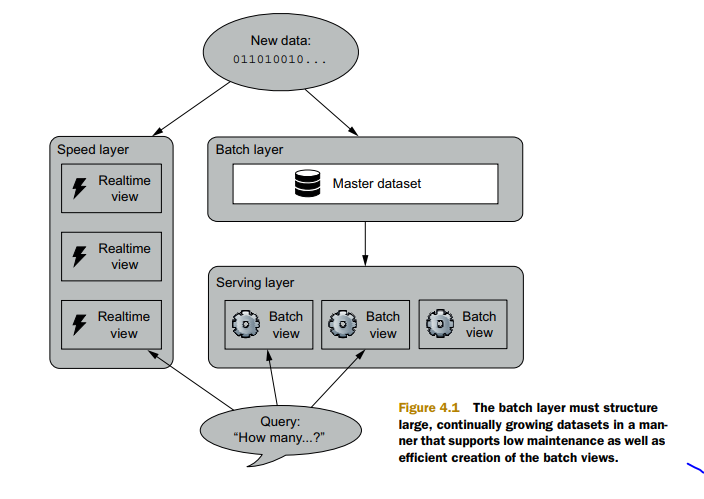
En el siguiente capítulo, aprenderá cómo almacenar físicamente un conjunto de datos maestro en el

Capa de proceso por lotes para que pueda procesarse de manera fácil y eficiente.

4. Data storage on the batch layer

En los últimos dos capítulos, aprendió sobre un modelo de datos para el conjunto de datos maestro y Cómo puedes traducir ese modelo de datos en un esquema gráfico. Viste la importancia De hacer que los datos sean inmutables y eternos. El siguiente paso es aprender físicamente almacenar esos datos en la capa de lote. La Figura 4.1 resume en dónde estamos Arquitectura lambda.

Al igual que los dos últimos capítulos, este capítulo está dedicado al conjunto de datos maestro. los el conjunto de datos maestro suele ser demasiado grande para existir en un solo servidor, por lo que debe elegir Cómo distribuirá sus datos en múltiples máquinas. La forma en que almacena su El conjunto de datos maestro afectará la forma en que lo consumes, por lo que es vital diseñar tu almacenamiento. estrategia con sus patrones de uso en mente.



En este capítulo harás lo siguiente:

■ Determinar los requisitos para almacenar el conjunto de datos maestro

■ Vea por qué los sistemas de archivos distribuidos son un ajuste natural para almacenar un conjunto de datos maestro

■ Vea cómo el almacenamiento de capa por lotes para el proyecto SuperWebAnalytics.com se asigna a sistemas de archivos distribuidos

Comenzaremos examinando cómo el rol de la capa de lotes dentro de la Arquitectura Lambda afecta cómo debes almacenar tus datos.

4.1 Requisitos de almacenamiento para el conjunto de datos maestro

Para determinar los requisitos para el almacenamiento de datos, debe considerar cómo sus datos se

Se escribirá y cómo se leerá. El papel de la capa de lotes dentro de la Lambda La arquitectura afecta a ambas áreas. Discutiremos cada una a un alto nivel antes de proporcionar una Lista completa de requisitos.

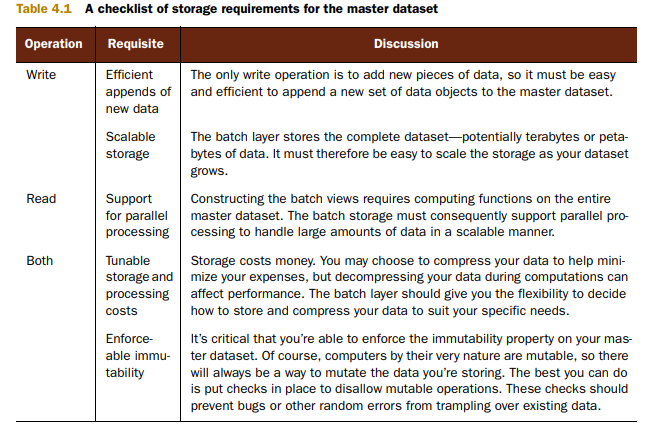
 En el capítulo 2 enfatizamos dos propiedades clave de los datos: los datos son inmutables y eternamente

cierto. En consecuencia, cada parte de sus datos se escribirá una sola vez. No es necesario modificar nunca sus datos, la única operación de escritura será agregar una nueva Unidad de datos a su conjunto de datos. Por lo tanto, la solución de almacenamiento debe optimizarse para manejar

Un gran conjunto de datos en constante crecimiento.

La capa de lotes también es responsable de las funciones de cálculo en el conjunto de datos para producir Las vistas por lotes. Esto significa que el sistema de almacenamiento de capa por lotes debe ser bueno en leyendo muchos datos a la vez. En particular, el acceso aleatorio a datos individuales es no requerido.

Con este paradigma de "escribir una vez, leer masivamente muchas veces" en mente, podemos crear un Lista de verificación de requisitos para el almacenamiento de datos: consulte la tabla 4.1.



Veamos ahora una clase de tecnologías que cumplen con estos requisitos.

4.3 Elegir una solución de almacenamiento para la capa de proceso por lotes

Con la lista de verificación de requisitos en la mano, ahora puede considerar opciones para lotes almacenamiento de capa. Con tales requisitos sueltos, que ni siquiera necesitan acceso aleatorio a la

Datos: parece que podría usar prácticamente cualquier base de datos distribuida para el maestro conjunto de datos Así que primero consideremos la viabilidad de usar un almacén de clave / valor, el más común Tipo de base de datos distribuida, para el conjunto de datos maestro.

4.2.1 Usando un almacén de clave / valor para el conjunto de datos maestro

No hemos discutido las tiendas de valor / clave distribuidas todavía, pero esencialmente puedes pensar en como gigantescos hashmaps persistentes que se distribuyen entre muchas máquinas. Si estás almacenando un conjunto de datos maestro en un almacén de clave / valor, lo primero que debes averiguar fuera es lo que deben ser las claves y cuáles deben ser los valores.

Lo que debe ser un valor es obvio: es un dato que desea almacenar, pero lo que ¿Debe ser una clave? No hay una clave natural en el modelo de datos, ni una es necesaria porque los datos deben consumirse en forma masiva. Así que inmediatamente golpeas un desajuste de impedancia entre el modelo de datos y cómo funcionan los almacenes clave / valor. El único realmente viable. La idea es generar un UUID para usar como clave.

Pero esto es solo el comienzo de los problemas con el uso de almacenes clave / valor para un maestro conjunto de datos Debido a que los almacenes de clave / valor necesitan acceso detallado a los pares de clave / valor para hacer aleatoriamente lee y escribe, no puede comprimir varios pares clave / valor juntos. Asi que eres gravemente limitado en el ajuste de la compensación entre los costos de almacenamiento y los costos de procesamiento.

 Las tiendas de clave / valor están destinadas a ser usadas como tiendas mutables, lo cual es un problema si imponer la inmutabilidad es tan crucial para el conjunto de datos maestro. A menos que modifiques el código. del almacén de clave / valor que está utilizando, normalmente no puede desactivar la capacidad de modificar Pares clave / valor existentes.

Sin embargo, el mayor problema es que una tienda de clave / valor tiene muchas cosas que usted no conoce. necesidad: lecturas aleatorias, escrituras aleatorias y toda la maquinaria detrás de hacerlas trabajo. De hecho, la mayor parte de la implementación de un almacén de clave / valor está dedicada a estos características que no necesitas en absoluto. Esto significa que la herramienta es enormemente más compleja que tiene que ser para cumplir con sus requisitos, por lo que es mucho más probable que tenga un problema con eso Además, el almacén de clave / valor indexa sus datos y proporciona

Servicios innecesarios, que aumentarán sus costos de almacenamiento y disminuirán su rendimiento

Al leer y escribir datos.

4.2.2 Sistemas de archivos distribuidos

Resulta que hay un tipo de tecnología con la que ya estás familiarizado. es un ajuste perfecto para el almacenamiento de capas por lotes: sistemas de archivos.

 Los archivos son secuencias de bytes, y la forma más eficiente de consumirlos es escaneando

a través de ellos. Se almacenan secuencialmente en el disco (a veces se dividen en bloques, pero la lectura y la escritura sigue siendo esencialmente secuencial). Tienes el control total sobre los bytes de un archivo, y tiene total libertad para comprimirlos sin embargo querer. A diferencia de un almacén de clave / valor, un sistema de archivos le brinda exactamente lo que necesita y no más, al tiempo que no limita su capacidad para ajustar el costo de almacenamiento en comparación con el costo de procesamiento.

Además de eso, los sistemas de archivos implementan sistemas de permisos específicos, que son perfectos Para imponer la inmutabilidad.

El problema con un sistema de archivos normal es que existe en una sola máquina, por lo que solo puede escalar hasta los límites de almacenamiento y la capacidad de procesamiento de esa máquina. Pero se vuelve Hay una clase de tecnologías llamadas sistemas de archivos distribuidos que es bastante similar a los sistemas de archivos con los que está familiarizado, excepto que distribuyen su almacenamiento en un grupo de ordenadores. Se escalan agregando más máquinas al clúster. Sistemas de archivos distribuidos están diseñados para que tenga tolerancia a fallos cuando una máquina se apaga, lo que significa que Si pierde una máquina, todos sus archivos y datos seguirán siendo accesibles.

Hay algunas diferencias entre los sistemas de archivos distribuidos y los sistemas de archivos regulares.

Las operaciones que puede hacer con un sistema de archivos distribuido a menudo son más limitadas

lo que puedes hacer con un sistema de archivos regular. Por ejemplo, es posible que no pueda escribir en

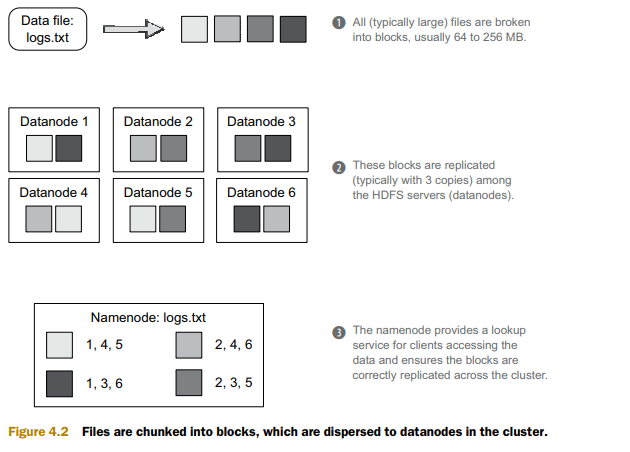
la mitad de un archivo o incluso modificar un archivo después de la creación. La mayoría de las veces, tener archivos pequeños puede ser ineficiente, por lo que debe asegurarse de mantener el tamaño de sus archivos relativamente grandes para hacer un uso adecuado del sistema de archivos distribuido (los detalles dependen de la herramienta, pero 64 MB es una buena regla de oro).

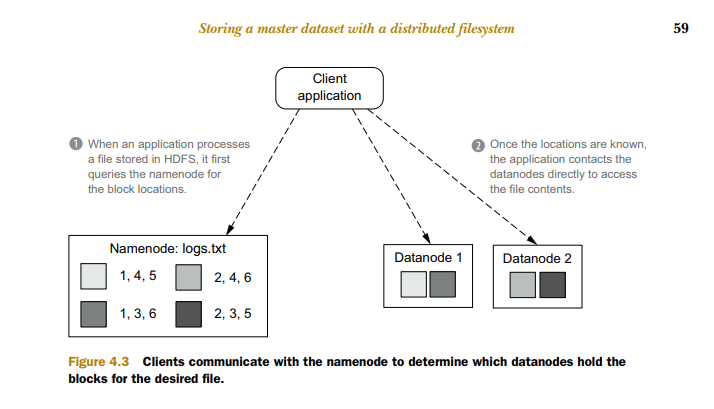
4.3 Cómo funcionan los sistemas de archivos distribuidos

Cómo funcionan los sistemas de archivos distribuidos Es difícil hablar en el resumen sobre cómo funciona cualquier sistema de archivos distribuido, por lo que fundamenta nuestra explicación con una herramienta específica: el Sistema de archivos distribuidos de Hadoop (HDFS). Creemos que el diseño de HDFS es lo suficientemente representativo de cómo los sistemas de archivos distribuidos trabaje para demostrar cómo se puede utilizar una herramienta de este tipo para la capa de lotes.

HDFS y Hadoop MapReduce son las dos puntas del proyecto Hadoop: un Java Marco para el almacenamiento distribuido y el procesamiento distribuido de grandes cantidades de datos. Hadoop se implementa en múltiples servidores, generalmente llamado clúster, y HDFS se implementa un sistema de archivos distribuido y escalable que administra la forma en que los datos se almacenan en el clúster. Hadoop es un proyecto de gran tamaño y profundidad, por lo que solo proporcionaremos un alto nivel descripción.

En un clúster HDFS, hay dos tipos de nodos: un único namenode y múltiples datanodes. Cuando carga un archivo a HDFS, el archivo se divide primero en bloques de una Tamaño fijo, normalmente entre 64 MB y 256 MB. Cada bloque se replica a través de múltiples datanodes (típicamente tres) que son elegidos al azar. El namenode mantiene seguimiento de la asignación de archivo a bloque y dónde se encuentra cada bloque. Este diseño es mostrado en la figura 4.2.





Distribuir un archivo de esta manera en muchos nodos permite que se procese fácilmente en paralelo. Cuando un programa necesita acceder a un archivo almacenado en HDFS, entra en contacto con el namenode para determinar qué datanodes alojan el contenido del archivo. Este proceso se ilustra en la figura. 4.3.

Además, con cada bloque replicado en múltiples nodos, sus datos permanecen disponible incluso cuando los nodos individuales están fuera de línea. Por supuesto, hay límites para esto. tolerancia a fallos: si tiene un factor de replicación de tres, tres nodos bajan a la vez, y estás almacenando millones de bloques, es probable que existan algunos bloques en exactamente esos tres nodos y no estará disponible.

Implementar un sistema de archivos distribuido es una tarea difícil, pero ahora has aprendido Lo que es importante desde la perspectiva del usuario. Para resumir, estos son los importantes. cosas que saber:

■ Los archivos se distribuyen en varias máquinas para ofrecer escalabilidad y también para habilitarlos en paralelo. tratamiento.

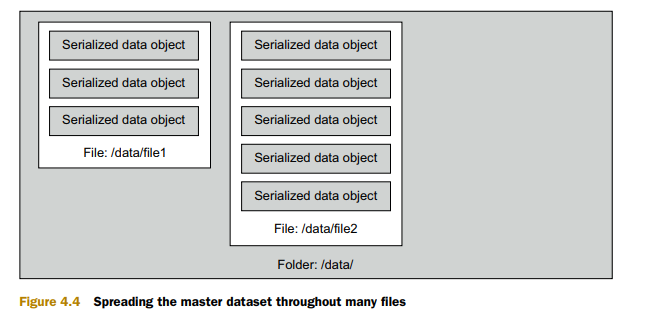
■ Los bloques de archivos se replican en varios nodos para la tolerancia a fallas.

Exploremos ahora cómo almacenar un conjunto de datos maestro utilizando un sistema de archivos distribuido.

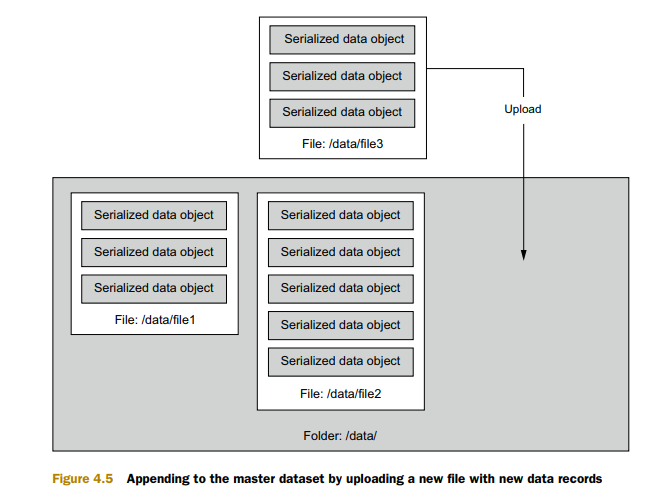
4.4 Almacenamiento de un conjunto de datos maestro con un sistema de archivos distribuido

Los sistemas de archivos distribuidos varían en los tipos de operaciones que permiten. Algunos distribuidos Los sistemas de archivos le permiten modificar los archivos existentes, y otros no. Algunos te permiten adjuntar a archivos existentes, y algunos no tienen esa característica. En esta sección veremos cómo puede almacenar un conjunto de datos maestro en un sistema de archivos distribuido con solo el más deshuesado de características, donde un archivo no puede ser modificado en absoluto después de ser creado.

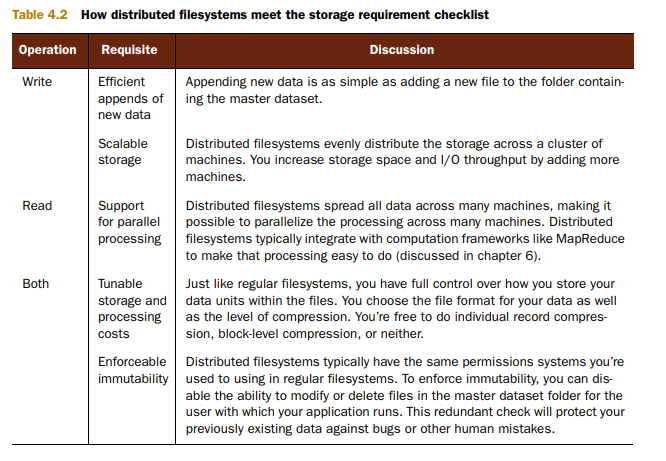
Claramente, con archivos no modificables no puede almacenar todo el conjunto de datos maestro en un solo expediente. Lo que puede hacer en su lugar es difundir el conjunto de datos maestro entre muchos archivos y almacenar todos esos archivos en la misma carpeta. Cada archivo contendría muchos objetos de datos serializados, como se ilustra en la figura 4.4.



Para adjuntarlo al conjunto de datos maestro, simplemente agregue un nuevo archivo que contenga los nuevos datos objetos a la carpeta del conjunto de datos maestro, como se muestra en la figura 4.5.



Revisemos ahora los requisitos para el almacenamiento del conjunto de datos maestro y verifiquemos que un sistema de archivos coincide con esos requisitos. Esto se muestra en la tabla 4.2.



A un alto nivel, los sistemas de archivos distribuidos son sencillos y un ajuste natural para el maestro conjunto de datos Por supuesto, como cualquier herramienta, tienen sus peculiaridades, y estas se discuten en El siguiente capítulo de ilustración. Pero resulta que hay un poco más que puedes explotar con la abstracción de archivos y carpetas para mejorar el almacenamiento del conjunto de datos maestro, así que ahora vamos a hablar sobre el uso de carpetas para habilitar la partición vertical.

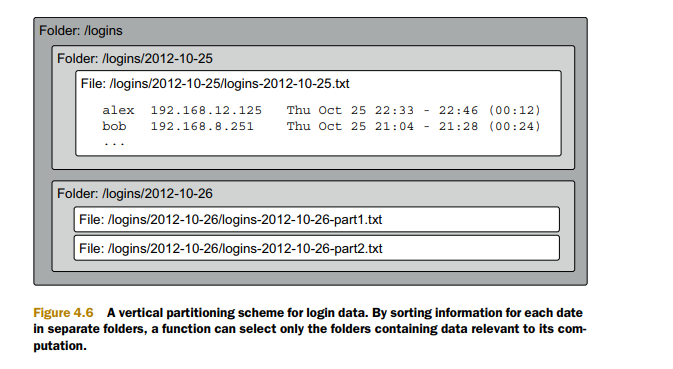
4.5 Particionamiento vertical

Aunque la capa de proceso por lotes está diseñada para ejecutar funciones en todo el conjunto de datos, muchos cálculos No es necesario mirar todos los datos. Por ejemplo, puede tener un cálculo

Eso solo requiere información recolectada durante las últimas dos semanas. El lote el almacenamiento debería permitirle particionar sus datos de modo que una función solo acceda a los datos Relevante a su computación. Este proceso se denomina partición vertical y puede Contribuir a hacer la capa de lote más eficiente. Si bien no es estrictamente necesario para la capa de lotes, ya que la capa de lotes es capaz de ver todos los datos a la vez y filtrando lo que no necesita, la partición vertical permite un gran rendimiento ganancias, por lo que es importante saber cómo usar la técnica.

La partición vertical de los datos en un sistema de archivos distribuido puede clasificarse su datos en carpetas separadas. Por ejemplo, supongamos que está almacenado información de inicio de sesión en un sistema de archivos distribuido. Cada inicio de sesión contiene un nombre de usuario, dirección IP y marca de tiempo. UNA Partición vertical por día, puede crear una carpeta separada para cada día de datos. Cada día Los archivos de archivos se publican en el sitio. Esto se ilustra en la figura. 4.6.

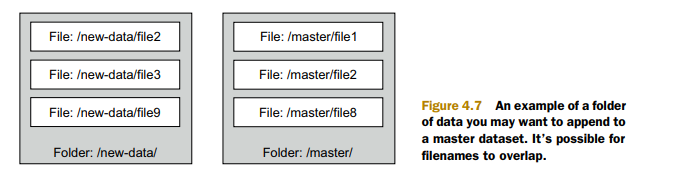
Ahora, si solo desea ver un subconjunto particular de su conjunto de datos, puede simplemente mirar en los archivos en esas carpetas particulares e ignora los otros archivos.



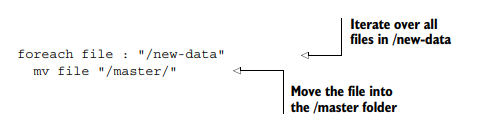
4.6 Naturaleza de bajo nivel de los sistemas de archivos distribuidos.

Mientras que los sistemas de archivos distribuidos proporcionan las propiedades de almacenamiento y tolerancia a fallos que necesidad de almacenar un conjunto de datos maestro, encontrará que el uso de sus API directamente es demasiado bajo para Las tareas que necesitas ejecutar. Ilustraremos esto utilizando las operaciones regulares del sistema de archivos Unix. y muestra las dificultades que puedes encontrar al realizar tareas, como añadirlas a un maestro conjunto de datos o partición vertical de un conjunto de datos maestro.

Comencemos con la adición a un conjunto de datos maestro. Supongamos que su conjunto de datos maestro está en la carpeta / master y usted tiene una carpeta de datos en / new-data que desea poner dentro de su conjunto de datos maestro. Supongamos que los datos en las carpetas están contenidos en archivos, como mostrado en la figura 4.7.



Lo más obvio de intentar es algo como el siguiente pseudocódigo:



Desafortunadamente, este código tiene serios problemas. Si la carpeta del conjunto de datos maestro contiene Cualquier archivo del mismo nombre, entonces la operación mv fallará. Para hacerlo correctamente, tienes para asegurarse de cambiar el nombre del archivo a un nombre de archivo aleatorio y evitar conflictos.

Hay otro problema. Uno de los requisitos básicos de almacenamiento para el maestro. conjunto de datos es la capacidad de ajustar las compensaciones entre los costos de almacenamiento y los costos de procesamiento. Al almacenar un conjunto de datos maestro en un sistema de archivos distribuido, elige un formato de archivo y Formato de compresión que hace la compensación que deseas. ¿Qué pasa si los archivos en / new-data ¿Son de un formato diferente que en / master? Entonces la operación de mv no funcionará en absoluto, usted en su lugar, debe copiar los registros de / new-data y en un archivo nuevo con el archivo formato utilizado en / master.

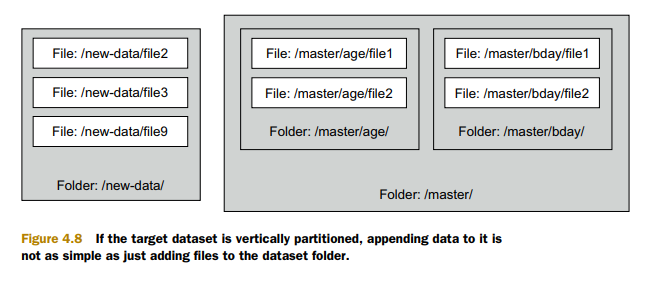
Veamos ahora la misma operación pero con una partición vertical conjunto de datos maestro. Supongamos que ahora / new-data y / master se parecen a la figura 4.8.

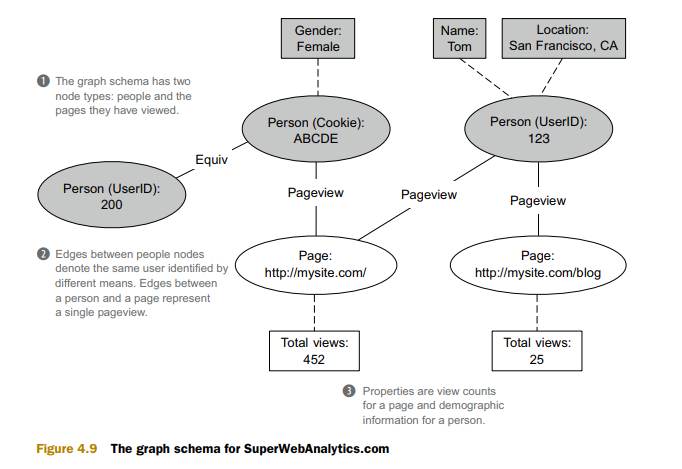
Simplemente poner los archivos de / new-data en la raíz de / master es incorrecto porque no respetaría la partición vertical de / master. O bien la operación de añadir debe ser rechazado, porque / new-data no está correctamente dividido verticalmente, o / new-data se debe particionar verticalmente como parte de la operación de adición. Pero cuando solo estás usando una API de archivos y carpetas directamente, es muy fácil cometer un error y romper las restricciones de partición vertical de un conjunto de datos.

Todas las operaciones y comprobaciones que deben realizarse para que estas operaciones funcionen.

correctamente indican fuertemente que los archivos y carpetas son demasiado bajos de una abstracción para manipulando conjuntos de datos. En el siguiente capítulo de ilustración, verá un ejemplo de un

Biblioteca que automatiza estas operaciones.





4.7 Storing the SuperWebAnalytics.com master dataset on a distributed filesystem

Veamos ahora cómo puede hacer uso de un sistema de archivos distribuido para almacenar el archivo maestro. conjunto de datos para SuperWebAnalytics.com.

La última vez que dejó este proyecto, creó un esquema gráfico para representar el conjunto de datos Cada borde y propiedad se representa a través de su propia DataUnit independiente. La Figura 4.9 resume cómo se ve el esquema del gráfico.

Una observación clave es que un esquema gráfico proporciona una partición vertical natural de los datos. Puede almacenar todos los tipos de borde y propiedad en sus propias carpetas. Partición vertical Los datos de esta manera le permiten ejecutar eficientemente cálculos que solo miran ciertos Propiedades y aristas.

4.8 summary

Los requisitos de alto nivel para almacenar datos en la capa de proceso por lotes de Lambda Architecture son sencillo. Usted observó que estos requisitos podrían ser asignados a un lista de verificación para una solución de almacenamiento, y vio que un sistema de archivos distribuido es un ajuste natural para este propósito. Usar y aplicar un sistema de archivos distribuido debe ser muy familiar.

En el siguiente capítulo verá cómo manejar los detalles esenciales del uso de un sistema de archivos en la práctica, y cómo lidiar con la naturaleza de bajo nivel de los archivos y Carpetas con una abstracción de nivel superior.