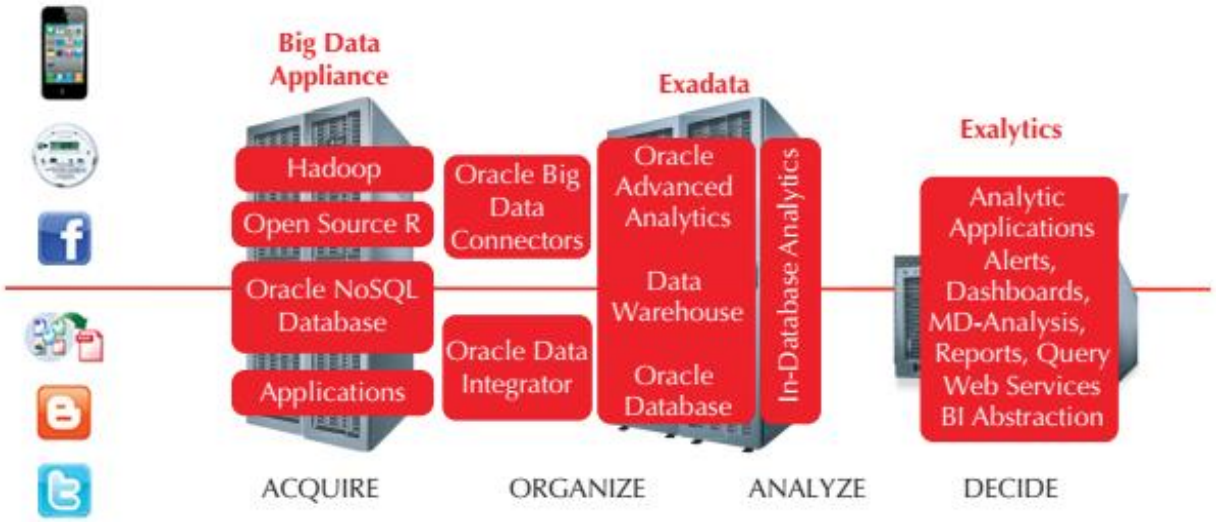
# Oracle noSQL

# Propósito

- Oracle NoSQL Database fue desarrollado para aumentar el producto Oracle Berkeley DB con escalabilidad horizontal elástica, una característica muy necesaria para las aplicaciones de Big Data, y para complementar la oferta de Big Data de Oracle.
- Con Oracle NoSQL Database, los datos se distribuyen automáticamente en varios servidores y se replican en un número configurable de estos servidores.
  - Los servidores se pueden agregar y quitar dinámicamente para adaptarse a los requisitos de administración de datos de una aplicación.
  - A medida que varía el número de servidores, Oracle NoSQL Database redistribuye los datos automáticamente para lograr el equilibrio de carga.
  - Los datos se redistribuyen simultáneamente, garantizando así un servicio continuo e ininterrumpido.
  - El rendimiento de las transacciones y la capacidad de datos de Oracle NoSQL Database escalan linealmente con el número de servidores.

- Oracle NoSQL Database utiliza Berkeley DB como su administrador de almacenamiento subyacente y lo aumenta con una capa de distribución de datos para la escalabilidad.
  - Por lo tanto, aprovecha las robustas propiedades ACID y la alta disponibilidad.
    - Oracle NoSQL Database ofrece un modelo de programación simple y soporte JSON.
  - Está integrado con Oracle Database y Hadoop, y es un componente base de Big Data Appliance de Oracle.



# Antecedentes

- El término "NoSQL" fue acuñado inicialmente por Carlo Strozzi en 1998 cuando nombró su sistema de gestión de bases de datos relacionales ligero y de código abierto como NoSQL.
- Aunque su base de datos todavía utilizaba el paradigma de la base de datos relacional, su intención principal era proporcionar una interfaz alternativa para el acceso a los datos además de SQL.
- El término "NoSQL" reapareció más tarde en 2009 como un intento de categorizar la gran cantidad de bases de datos emergentes que desafiaban los atributos de los sistemas RDBMS tradicionales.
- Los atributos clave de las bases de datos NoSQL son principalmente para soportar estructuras no relacionales; proporcionar una implementación distribuida que sea altamente escalable; y en la mayoría de las ocasiones, no admitir las características clave de garantía de transacción inherentes a los sistemas RDBMS, como las propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad).

- Son capaces de proporcionar un rendimiento rápido en las escrituras porque utilizan un modelo de datos simple en el que los datos se almacenan tal cual con su estructura original, junto con una sola clave de identificación, en lugar de interpretar y convertir los datos en un esquema bien definido.
- Las lecturas también se vuelven muy simples: proporciona una clave y la base de datos devuelve rápidamente el valor realizando una búsqueda de índice basada en claves.
- Las bases de datos NoSQL también se distribuyen y replican para proporcionar alta disponibilidad y confiabilidad, y pueden escalar linealmente en rendimiento y capacidad simplemente agregando más nodos de almacenamiento al clúster.
- Con esta arquitectura ligera y distribuida, las bases de datos NoSQL pueden almacenar rápidamente una gran cantidad de transacciones y proporcionar búsquedas extremadamente rápidas.

# Beneficios

- La función de alta disponibilidad y replicación de Berkeley DB permite que una aplicación sobreviva a los fallos de la máquina y mejore la escalabilidad de lectura.
- Una aplicación Berkeley DB de alta disponibilidad se ejecuta en varios equipos configurados como un clúster de alta disponibilidad; las actualizaciones de la base de datos solo se permiten en un equipo, designado como maestro.
- La aplicación que se ejecuta en los otros nodos (llamadas réplicas) puede leer los datos.
- Berkeley DB propaga los cambios en los datos del nodo maestro a todas las réplicas de las demás máquinas del clúster para mantener las réplicas actualizadas y actualizadas.
- Si la máquina que ejecuta el maestro falla, Berkeley DB proporciona un mecanismo de elección que se puede usar para elegir un nuevo maestro de entre las réplicas sobrevivientes sin interrupción de la actividad normal.

- Oracle NoSQL Database es un sistema de nada compartido diseñado para ejecutarse y escalar en hardware básico.
- Los pares clave-valor se particionan en hash entre grupos de servidores conocidos como particiones.
  - En cualquier momento, un único par clave-valor siempre está asociado con un fragmento único en el sistema.
  - La clave principal (descrita brevemente) del par clave-valor se hashea para determinar a qué fragmento pertenecerá el registro.
- La mayoría de las implementaciones de Oracle NoSQL Database utilizan varias máquinas (también denominadas nodos) por fragmento; cada fragmento se configura como un sistema de alta disponibilidad utilizando la función de alta disponibilidad de Berkeley DB Java Edition.
- La configuración recomendada requiere un mínimo de tres máquinas por fragmento; esto se denomina factor de replicación para la configuración.
- Dependiendo de los requisitos de la aplicación, un factor de replicación mayor o menor que 3 podría ser más apropiado.
  - Por ejemplo, un sistema de 10 particiones de alta disponibilidad con un factor de replicación de 3 se implementaría en 30 nodos. Por supuesto, otras configuraciones son posibles en la práctica.

# Características

- Oracle NoSQL Database admite la noción de claves menores. La combinación de claves principales y secundarias se puede utilizar para identificar y abordar partes específicas de la información en un registro de par clave-valor.
  - Las claves menores son opcionales, pero proporcionan una comodidad significativa al desarrollador de la aplicación.
  - La combinación de teclas mayores y menores sirve como la clave primaria única totalmente calificada.
- Oracle NoSQL Database proporciona APIs para acceder a todo el contenido de un registro de par clave-valor específico, así como APIs para acceder a partes del registro identificadas por una combinación de claves principal y menor.

Una operación en Oracle NoSQL Database solo puede afectar al contenido de una única clave principal.

Además, las operaciones de Oracle NoSQL Database son transacciones de llamada de una sola API (excepto para escanear el contenido de toda la base de datos) donde cada solicitud de API del cliente al servidor es una unidad atómica de trabajo.

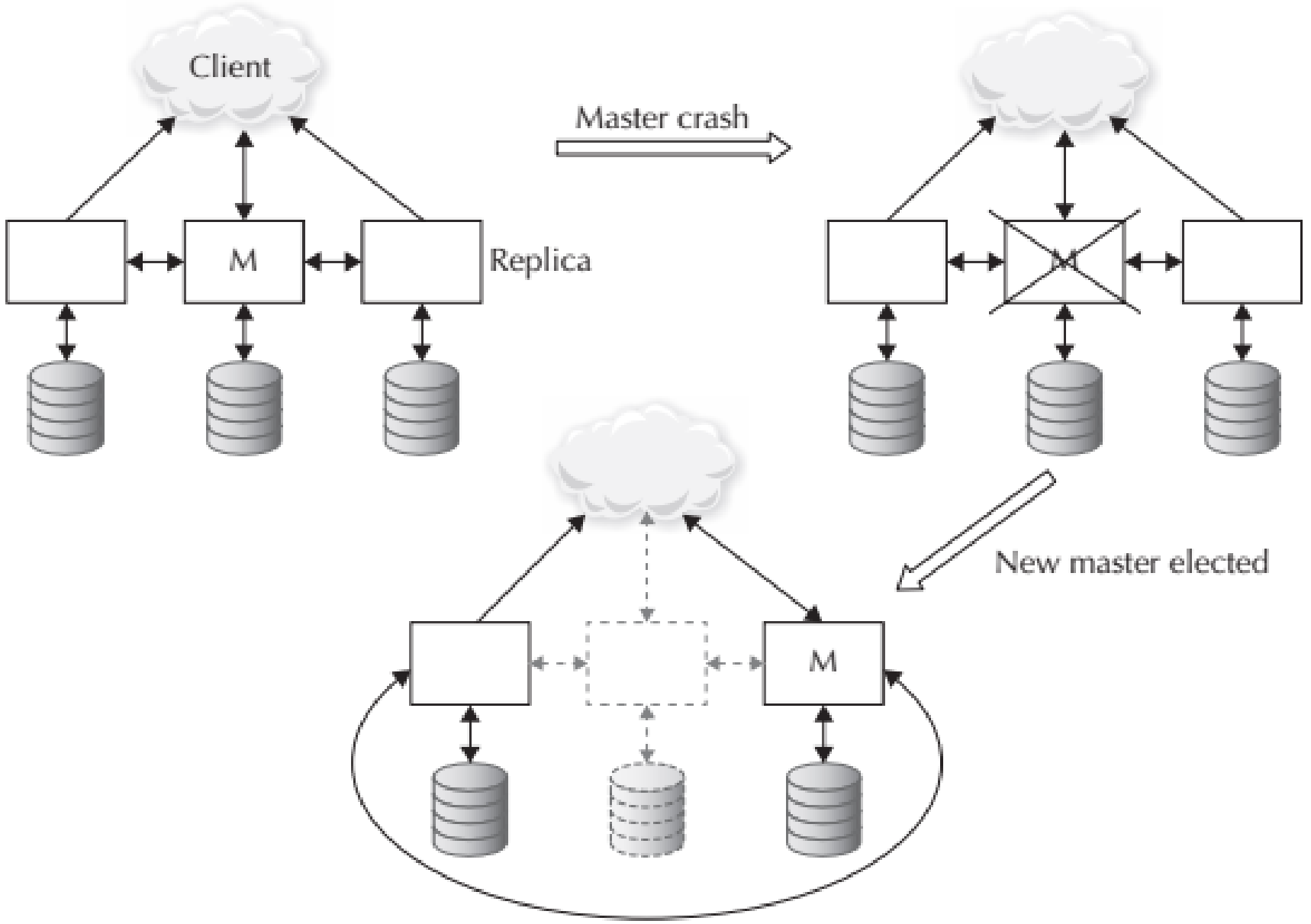
- En el contexto de una única clave principal, una solicitud de cliente puede modificar el contenido de algunas claves secundarias, eliminar otras y agregar algunas claves secundarias nuevas (y valores); todas estas actividades se ejecutan como una sola transacción.
- El sólido soporte de transacciones ACID es una de las características distintivas clave de Oracle NoSQL Database.

En caso de fallo del nodo, Oracle NoSQL Database gestiona las elecciones de forma automática y transparente a la aplicación.

Aparte de un retraso momentáneo mientras la elección está en curso, la aplicación no se ve afectada por fallas de nodo.

Además, Oracle NoSQL Database optimiza automáticamente la colocación de maestros y réplicas en los servidores de hardware para garantizar el mejor rendimiento del sistema.

- Existen dos enfoques alternativos para manejar las actualizaciones de datos en un fragmento de alta disponibilidad.
  - Un enfoque es designar una de las réplicas como el nodo maestro; un maestro puede servir solicitudes de actualización, así como solicitudes de lectura, mientras que todos los demás nodos solo pueden servir solicitudes de lectura. Esta arquitectura se llama *single-master*.
  - Otro enfoque es permitir actualizaciones en cualquier nodo del fragmento y luego propagar esos cambios a las otras réplicas. Esta arquitectura se llama *multi-master*
- La ventaja de una arquitectura de maestro único es que no puede haber cambios simultáneos en el mismo registro en varias réplicas; el maestro siempre tiene el valor más actual de cualquier registro en la partición.
- Un sistema de maestro único debe tener un mecanismo para elegir un nuevo maestro de una de las réplicas sobrevivientes en el fragmento, en caso de que el maestro actual falle.
  - La reelección maestra utiliza un algoritmo distribuido basado en quórum para elegir inequívocamente un nuevo maestro. Esta es la razón por la que la mayoría de los sistemas de alta disponibilidad tienen tres (o un número impar de) réplicas; esto asegura que sea posible reunir una mayoría de votos para determinar correctamente el resultado de una elección maestra.
  - La elección de un nuevo maestro suele ser un proceso muy rápido, que no dura más de un segundo o dos; durante este período, la actividad de actualización de la partición se suspende temporalmente. .


Client
M
Replica
Master crash
M
New master elected
M

- Cuando una aplicación realiza un cambio en un elemento de datos en un equipo, ese cambio debe propagarse a las otras réplicas. Debido a que la propagación del cambio no es instantánea, hay un intervalo de tiempo durante el cual algunas de las copias tendrán el cambio más reciente, pero otras no.
- Sin embargo, el cambio eventualmente se propagará a todas las copias.
  - En un sistema de maestro único, si una aplicación realiza un cambio en un registro, esa solicitud será manejada por el nodo maestro.
  - Tan pronto como se complete la solicitud de actualización, si la aplicación recupera el mismo registro (la misma clave principal), es posible que la solicitud se envíe a una de las réplicas del fragmento.
  - Si el maestro aún no ha propagado los cambios a esa réplica, la aplicación verá la versión anterior de los datos.
  - Sin embargo, si la aplicación solicita los datos después de que los cambios en el maestro se hayan propagado a la réplica, la aplicación verá la versión más reciente del registro.
  - Dependiendo del tiempo relativo de la solicitud de lectura, la aplicación puede ver diferentes valores
- La noción de consistencia eventual es simplemente un reconocimiento de que hay un retraso ilimitado en la propagación de un cambio realizado en una máquina a todas las demás copias.

Varios sistemas distribuidos abordan la consistencia de diferentes maneras porque existe una compensación entre la latencia de la operación, la disponibilidad y la consistencia.

En algunos sistemas, la máquina donde se origina el cambio simplemente enviará mensajes asincrónicos (y posiblemente no confiables) a las otras máquinas y declarará que la operación es exitosa.

- Esto es rápido, pero a costa de una posible pérdida de datos si la máquina de origen falla antes de que las réplicas hayan recibido la actualización.

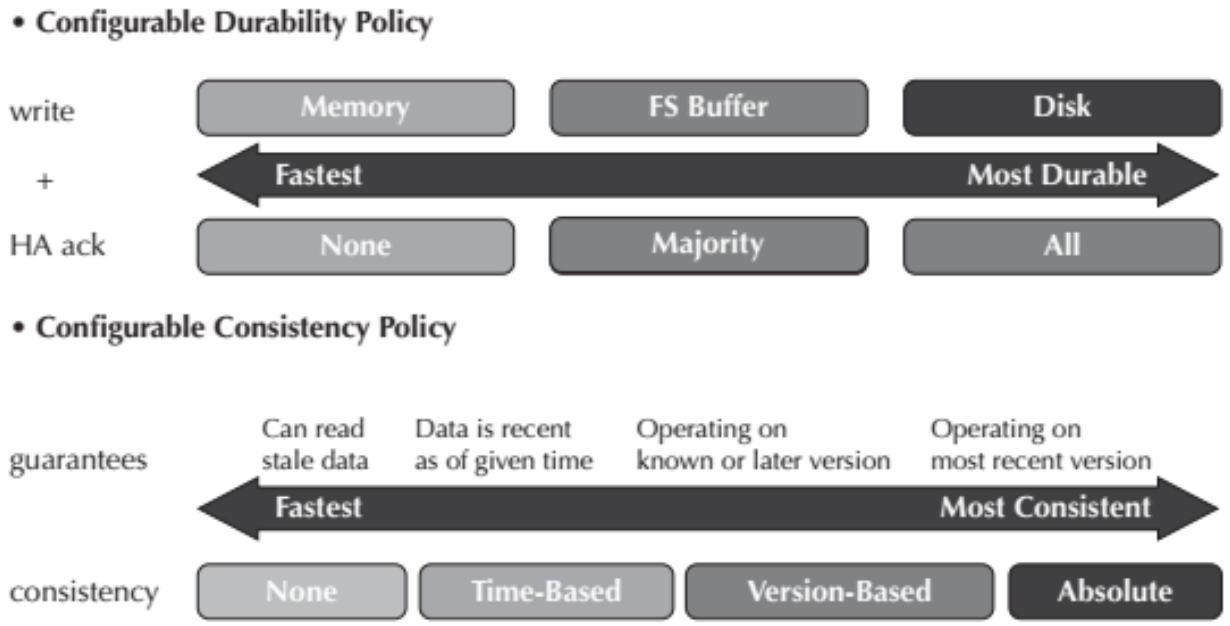
Otros sistemas envían mensajes sincrónicos (de bloqueo) a todas las demás máquinas, reciben confirmaciones y, solo entonces, declaran que la operación es exitosa.

- Estos sistemas favorecen la consistencia y la disponibilidad a costa del rendimiento.

Finalmente, un sistema podría implementar alguna variante de estos dos extremos (por ejemplo, esperar los reconocimientos de la mayoría de las réplicas).

- Oracle NoSQL Database ofrece varias opciones para la consistencia de lectura.
  - La aplicación puede especificar la coherencia absoluta si necesita la versión más reciente; en este caso, el controlador cliente enrutará la solicitud al nodo maestro de la partición.
  - El desarrollador de aplicaciones también puede especificar consistencia basada en el tiempo o en el ID de transacción para las operaciones de lectura
  - Algunos sistemas de bases de datos (y los sistemas NoSQL, en particular) han implementado una variedad de garantías de durabilidad relajadas para satisfacer las necesidades de dichas aplicaciones.
  - Por ejemplo, algunos sistemas amortiguan los cambios en la memoria y solo propagan los cambios al disco periódicamente. Un sistema puede optar por escribir el contenido del búfer en el disco cada 5 segundos.

- Sin embargo, si hay una falla (pérdida en memoria), se perderá el conjunto más reciente de cambios.
  - Otros sistemas pueden optar por emitir la E/S a los búferes del sistema operativo y declarar que el cambio es duradero antes de que el sistema operativo escriba los datos en búfer en el disco. En este caso, una falla del proceso (pero no una falla del sistema operativo) no afectará la durabilidad de los cambios; sin embargo, una falla del sistema operativo resultará en la pérdida de datos de los cambios más recientes.
- Por supuesto, el método más estricto (y más caro) para garantizar la durabilidad es emitir la E/S y luego esperar a que se complete la operación de escritura. También es posible escribir varios discos (por lo general, esto lo hace el sistema operativo o el subsistema de almacenamiento) para garantizar que los cambios también puedan sobrevivir a una falla del disco.
- El sistema puede declarar que una operación es duradera después de recibir confirmaciones de la actualización de las réplicas, sin esperar a que se complete la E/S de disco porque las réplicas han recibido la actualización (es duradera en otro nodo).




- Oracle NoSQL Database permite al usuario elegir la política de durabilidad por operación.
- Oracle NoSQL Database utiliza esta información durante el procesamiento de confirmaciones de transacciones para lograr el mejor rendimiento al tiempo que cumple con los requisitos de durabilidad de la operación.

# Claves principales, claves secundarias y valores

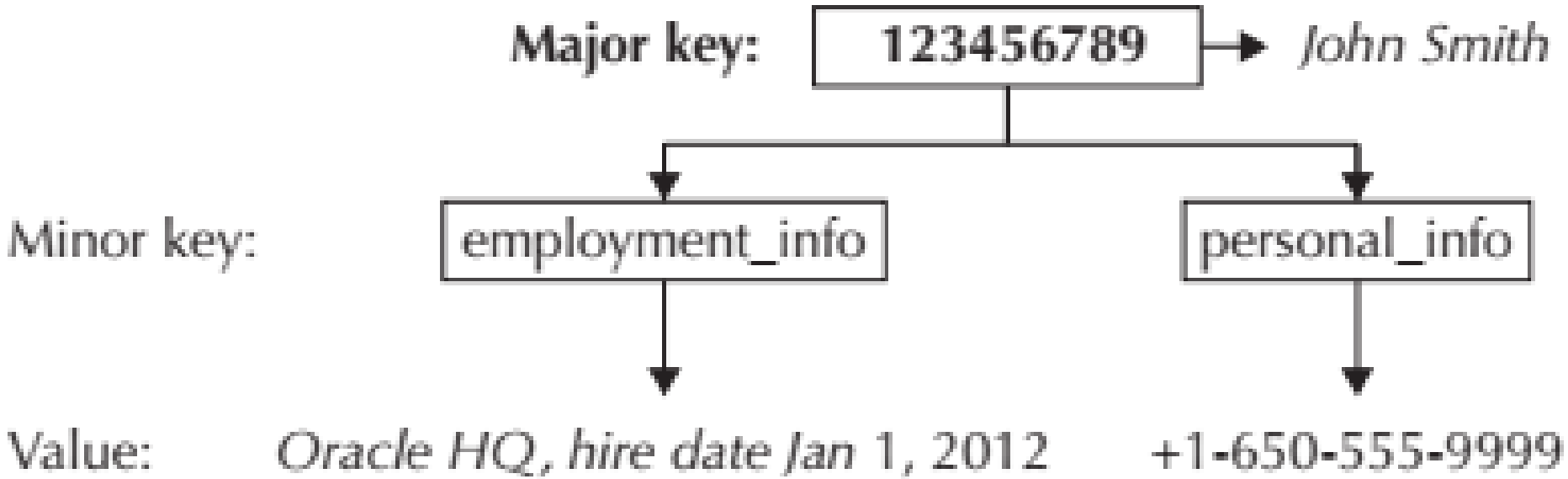
Cada entidad (registro) es un conjunto de pares clave-valor. Una clave tiene varios componentes, especificados como una lista ordenada.

La clave mayor identifica la entidad y consta de los componentes principales de la clave. Los componentes posteriores se denominan claves menores.

- Esta organización es similar a una especificación de ruta de directorio en un sistema de archivos (por ejemplo, /Mayor/menor1/menor2/). La parte "valor" del par clave-valor es simplemente una cadena no interpretada de bytes de longitud arbitraria.

La API para manipular pares clave-valor es simple.

- El usuario puede insertar un único par clave-valor en la base de datos mediante una operación put().
- Dada una clave, el usuario puede recuperar el par clave-valor mediante una operación get() o eliminarlo mediante una operación delete().
- Las operaciones get(), put() y delete() operan con una sola clave (multicomponente).




- La clave mayor determina a qué fragmento pertenecerá el registro.
- Todos los pares clave-valor asociados a la misma entidad (misma clave principal) se almacenan siempre en la misma partición.
  - Esta implementación permite un acceso eficiente y de un solo fragmento a subconjuntos del registro relacionados lógicamente.
  - Hay que tener en cuenta que las claves secundarias se pueden anidar

# Compatibilidad con objetos grandes

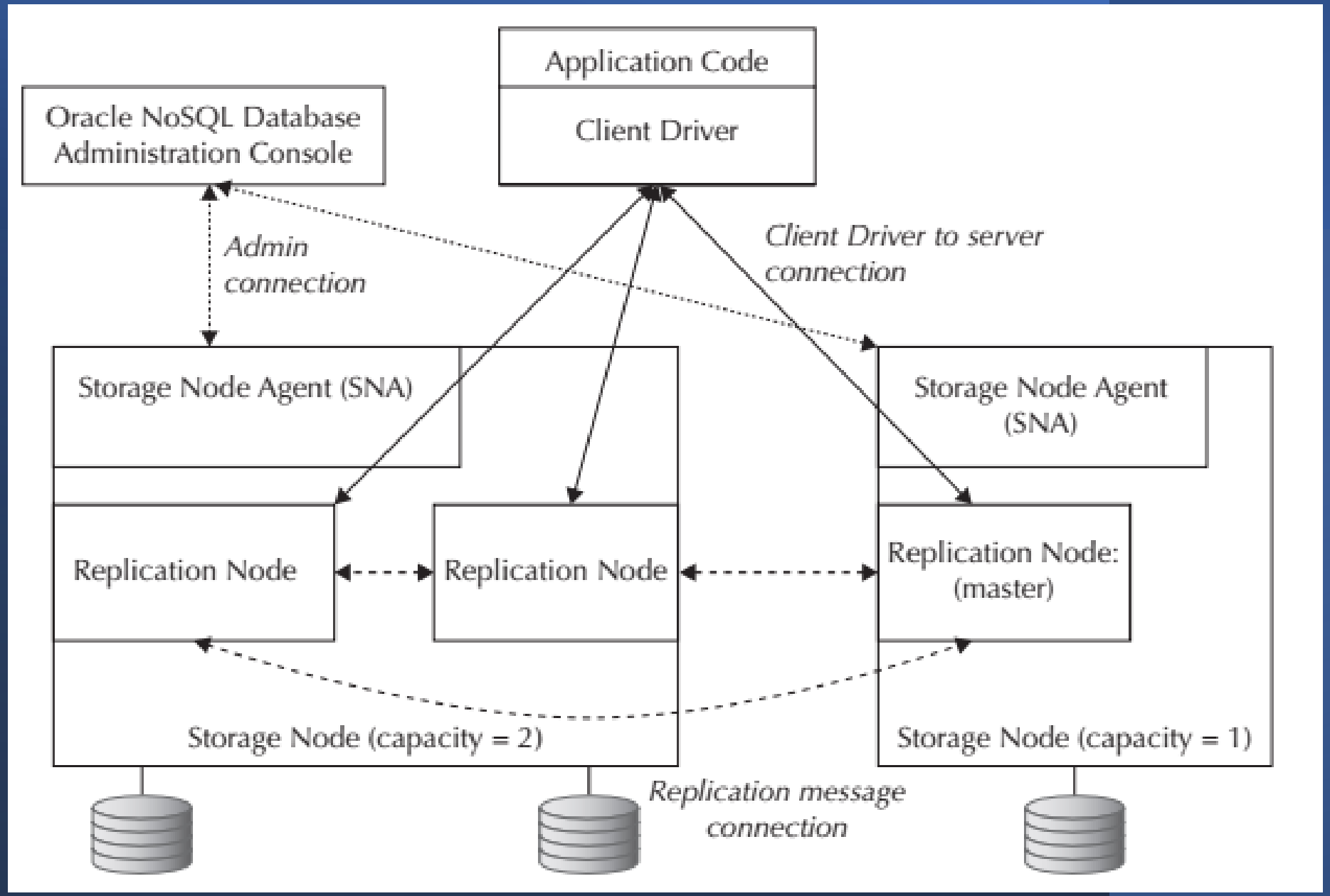
- Una base de datos Oracle NoSQL se utiliza a menudo para almacenar objetos grandes como imágenes, audio, vídeos y mapas.
- Un objeto grande se almacena internamente como una secuencia de fragmentos de objetos (o fragmentos).
- Además, la API de transmisión garantiza que los fragmentos se puedan obtener de manera eficiente de los fragmentos que los contienen.
- Oracle NoSQL Database administra datos de pares clave-valor; la clave y el valor pueden ser cadenas de byte arbitrarias que sólo son interpretadas por la aplicación, lo que puede dificultar el intercambio de datos entre varias aplicaciones.

# Esquemas JSON

- Oracle NoSQL Database también admite esquemas JSON (http:// json-schema.org/) y Apache Avro (http://avro.apache.org/) para especificar la estructura del valor en un par clave-valor.
  - Los esquemas JSON son autodescriptos, admiten la evolución de los esquemas y se utilizan ampliamente en aplicaciones de Big Data.
  - Apache Avro es un formato de serialización extremadamente eficiente en el espacio para esquemas JSON; por lo tanto, el uso de esquemas JSON y la serialización de Avro permiten facilitar el diseño de aplicaciones y el intercambio de datos entre varias aplicaciones y sistemas.
- Oracle NoSQL Database se utiliza a menudo para administrar datos web y de comercio electrónico. JSON y Javascript se utilizan popularmente en estas aplicaciones.

# Administración

- Oracle NoSQL Database incluye utilidades de administración para gestionar tareas operativas como configurar el sistema, definir la topología de la configuración del sistema y agregar nuevos recursos según sea necesario.
- También incluye herramientas de monitoreo para rastrear el estado del sistema en general, así como de los componentes individuales, detectar problemas de rendimiento y puntos críticos, y redistribuir dinámicamente el trabajo según sea necesario.
- Licenciamiento
- Oracle NoSQL Database se distribuye como una versión de código abierto, así como una versión empresarial. La **Community Edition** está disponible bajo la licencia de código abierto AGPLv3 y está diseñada para su uso en aplicaciones de código abierto.
- Oracle NoSQL Database **Enterprise Edition** está disponible bajo una licencia comercial y está destinado a aplicaciones propietarias.
  - Ambas versiones del producto proporcionan las mismas capacidades básicas que se necesitan para administrar grandes cantidades de datos clave-valor.
- **Enterprise Edition** también ofrece una integración más estrecha entre Oracle NoSQL Database y otros productos relacionados con Oracle, como RDF Graph, Oracle Event Processing, Oracle Coherence y Oracle Database.


Application Code
Client Driver
Oracle NoSQL Database
Administration Console
Admin
connection
Client Driver to server
connection
Storage Node Agent (SNA)
Storage Node Agent
(SNA)
Replication Node
Replication Node
Replication Node:
(master)
Storage Node (capacity = 2)
Storage Node (capacity = 1)
Replication message
connection

A diferencia de otros sistemas, Oracle NoSQL Database no tiene un repositorio central que mantenga la información de estado para todos los nodos en el servidor porque esto crearía un único punto de falla, así como un único punto de contención ("hotspot") en el sistema.

- En su lugar, cada controlador de cliente mantiene información sobre la asignación de claves a particiones para poder enrutar una solicitud de una clave especificada a la partición adecuada.
- Las claves se asignan a fragmentos mediante una función hash.

El controlador de cliente también realiza un seguimiento del estado de cada nodo de replicación en cada fragmento, incluida la información sobre qué nodo es actualmente el maestro, qué nodos son réplicas y qué nodos están sin conexión en un momento dado.

Debido a que las claves se distribuyen mediante un esquema hash, generalmente no hay una localidad de referencia entre las solicitudes de datos consecutivas; en consecuencia, cada cliente se comunica con frecuencia con cada nodo del servidor y se da cuenta de los cambios de estado muy pronto después de que se produce el cambio.

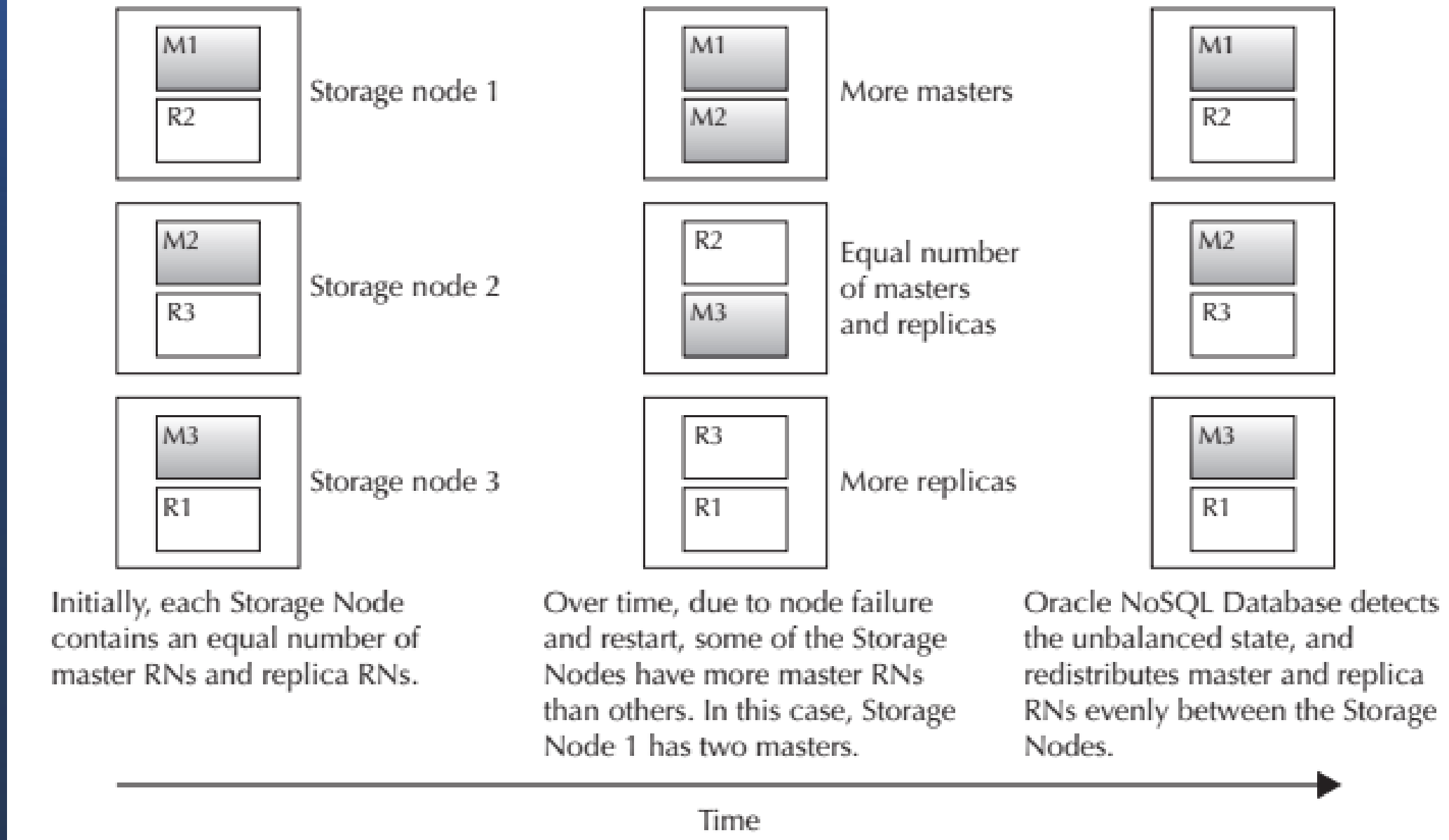
- Por lo tanto, si un nuevo cliente se conecta al servidor, también puede descubrir el estado de todos los nodos de replicación.

- Un cambio en un registro se propaga a las réplicas mediante el envío de los registros de bitácora (log).
  - Cada vez que se cambia un registro en el nodo maestro en un fragmento, se escribe en el almacenamiento local del maestro; en paralelo, el nodo maestro envía los registros de bitácora asociados con el cambio a cada una de las réplicas de la partición. Cuando estos registros de bitácora se reciben en la réplica, se aplican a la copia de los datos de la réplica para reflejar este cambio.
- En esencia, cada réplica está constantemente jugando a "ponerse al día" con respecto al nodo maestro.
  - Por lo tanto, dependiendo de la tasa de cambios y la latencia de la conexión de red entre el maestro y las réplicas, es posible que una réplica esté ligeramente desactualizada con respecto al maestro porque los registros de bitácora aún no se han recibido o aplicado en el lado de la réplica.
- Cada registro de bitácora se identifica con un número de secuencia de bitácora único y monótonamente creciente (*LSN Log Sequence Number*).

- Debido a que los LSN son monótonamente crecientes y únicos, el registro de bitácora para el cambio más reciente en cualquier momento tendrá el LSN más alto.
  - Cada vez que un nodo de replicación envía una respuesta al controlador de cliente, incluye el LSN más alto de su bitácora en el mensaje de respuesta.
  - En consecuencia, cada controlador de cliente puede realizar un seguimiento del LSN más alto para cada nodo de replicación en cada fragmento.
  - En particular, para cada fragmento, el cliente es consciente de si una réplica tiene el mismo o menor LSN que el nodo maestro; si el LSN de la réplica es inferior al del nodo maestro, también es consciente del "retraso" entre el nodo maestro y el nodo de replicación.
- Oracle NoSQL Database realiza un seguimiento de la asociación entre el tiempo y el LSN de un cambio en un registro. Por lo tanto, si una réplica se retrasa detrás del nodo maestro, es posible inferir el “hueco” de tiempo entre la réplica y el maestro, así como el retraso de LSN. Obviamente, el LSN de una réplica nunca puede ser más alto que el LSN del maestro.
- Para cada nodo del clúster, el controlador de cliente también realiza un seguimiento del número de solicitudes actualmente activas a cada nodo de replicación.
  - El controlador de cliente utiliza esta información como heurística para enrutar una solicitud de lectura a un nodo de replicación con la carga más ligera, si más de un nodo de replicación puede satisfacer las restricciones de coherencia de lectura de la operación.

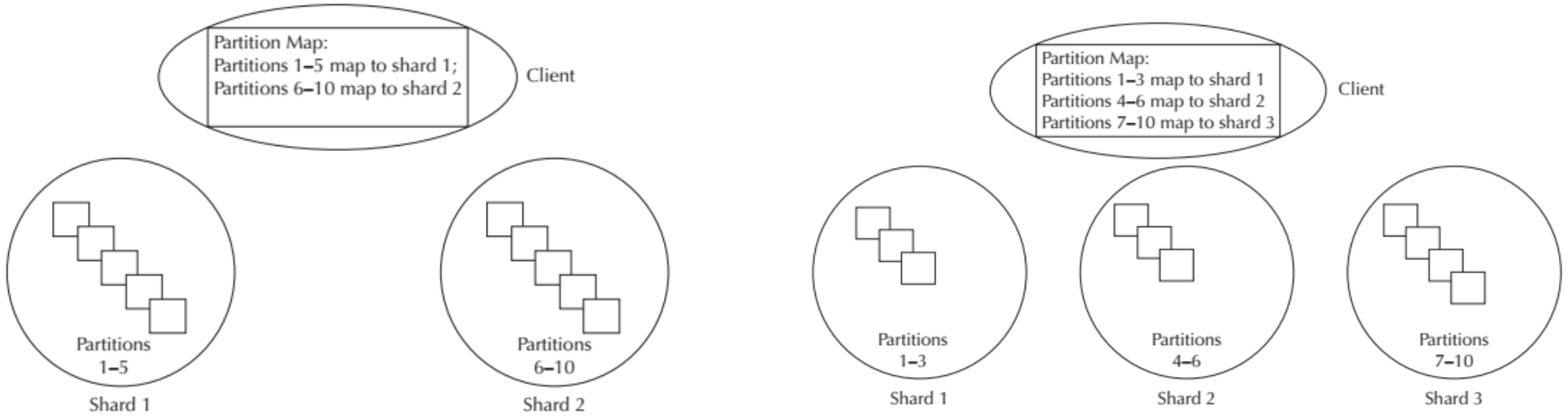
- Los nodos dentro de un fragmento están estrechamente acoplados.
  - Cada miembro del fragmento realiza un seguimiento del estado (en línea o fuera de línea) del otro miembros en la partición
  - Si uno de los nodos de una partición se desconecta, los nodos restantes participan en una elección para asegurarse de que hay un nodo maestro único en la partición.
  - Los fragmentos nunca se comunican entre sí, excepto durante las operaciones de redistribución (migración) de datos que permiten que el sistema escale linealmente a medida que se agregan más fragmentos (escalabilidad horizontal)
- Oracle NoSQL Database no requiere que todo el hardware del clúster sea idéntico. Algunos nodos de almacenamiento pueden tener más capacidad de procesamiento y almacenamiento que otros.
- Aunque la capacidad de procesamiento y la memoria también son consideraciones importantes, el ancho de banda de E/S disponible es la consideración más importante desde el punto de vista del rendimiento.

# Particiones, almacenamiento y topología de red


M1
R2
Storage node 1
M2
R3
Storage node 2
M3
R1
Storage node 3
M1
M2
More masters
R2
M3
Equal number of masters and replicas
R3
R1
More replicas
M1
R2
M2
R3
M3
R1
Initially, each Storage Node contains an equal number of master RNs and replica RNs.
Over time, due to node failure and restart, some of the Storage Nodes have more master RNs than others. In this case, Storage Node 1 has two masters.
Oracle NoSQL Database detects the unbalanced state, and redistributes master and replica RNs evenly between the Storage Nodes.
Time

# Hashing, particiones, distribución de datos

- Cada clave distinta (clave mayor) está asociada exactamente con un fragmento.
  - Oracle NoSQL Database utiliza un algoritmo basado en hash de dos etapas para asignar un par de valores clave a un fragmento
  - La clave mayor para un par clave-valor es una cadena Java.
  - El controlador de cliente hace un hash de la clave mayor mediante una función hash basada en MD5 para determinar un bucket hash.
- Oracle NoSQL Database utiliza el término partición para denotar un bucket hash. El número de particiones es fijo (hash estático) y definido por el administrador del sistema cuando se crea un nuevo store de Oracle NoSQL Database
- Oracle NoSQL Database asigna conjuntos de particiones de igual tamaño a cada nodo
  - Cada controlador de cliente mantiene un mapa de particiones, que contiene la asociación entre los ID de partición y las particiones.




- Suponiendo que el administrador del sistema agrega suficiente hardware para acomodar un fragmento más al clúster e inicia una operación de redistribución de datos.
  - Cuando se agrega un nuevo fragmento al clúster, Oracle NoSQL Database mueve algunas de las particiones de los fragmentos existentes al nuevo fragmento disponible para redistribuir los datos de manera uniforme entre el número total de fragmentos en el sistema.
  - Esta redistribución de datos se conoce como *migración de particiones*. Una operación de redistribución siempre mueve algunas particiones de cada fragmento existente

Debido a que las claves se hashean en las particiones disponibles, es posible que algunas particiones sean más grandes que otras.

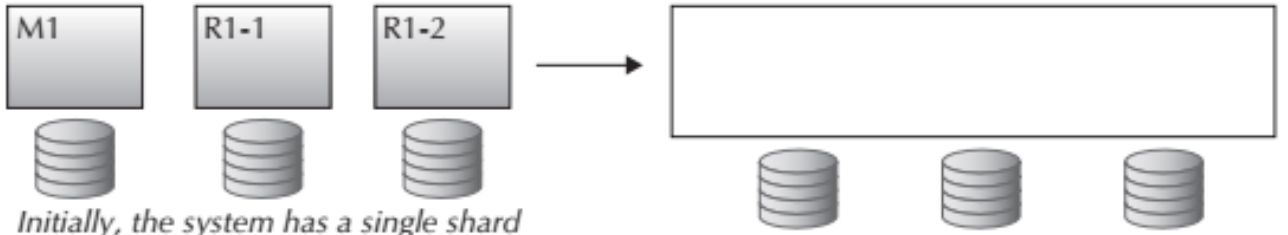
- Durante la operación de redistribución, Oracle NoSQL Database mueve particiones de tal manera que cada fragmento administra cantidades aproximadamente iguales de datos.
- Esto garantiza que cada fragmento maneje la misma carga de trabajo

El factor de replicación de un almacén de Oracle NoSQL Database define el número de réplicas que tiene cada fragmento. Cada fragmento dentro de un almacén de oracle NoSQL Database debe tener el mismo factor de replicación.

- Si la capacidad acumulada del hardware adicional no es un múltiplo del factor de replicación, se dejará sin usar parte de la capacidad.
- En el caso extremo, si no hay suficiente capacidad para alojar incluso un solo fragmento nuevo, Oracle NoSQL Database no redistribuirá los datos; el hardware recién asignado no será utilizable hasta que se agregue más capacidad.

La migración de particiones se realiza como una actividad en línea en segundo plano; el acceso a las aplicaciones continúa sin interrupción.

- Antes de comenzar el proceso de redistribución de particiones, el planificador de migración determina si se requiere tal movimiento.
- Si es necesario mover algunos nodos de replicación de un nodo de almacenamiento a otro, ese movimiento se inicia primero, antes de que se redistribuyan las particiones.




*Initially, the system has a single shard and each Replication Node is hosted on a Storage Node with capacity = 1.*

*Later, a new Storage Node with capacity of 3 is added. This means we can add one more shard.*


*The above configuration depicts the correct placement of new replicas and masters. No shard has all Replication Nodes on the same Storage Node.*


*The above configuration is ncorrect with respect to placement of Replication Nodes for the new shard. Shard 2 has all replication nodes on the same Storage Node. If that Storage Node is lost, the entire shard will become unavailable.*

- En cualquier momento dado durante una migración de partición, algunas filas de la partición seguirán estando en el fragmento original, otras filas estarán en tránsito y el resto de las filas se habrán transferido al nuevo fragmento.
- Para simplificar la coordinación y ejecución de la migración de particiones, solo los nodos maestros participan directamente en el proceso de migración.
- Para cada fragmento de origen, una vez que todas las filas de una partición se mueven al fragmento de destino, la partición se puede eliminar en el registro de fragmento de origen (maestro) garantizando que el cambio correspondiente también se produzca en las réplicas del fragmento de origen.
  - Del mismo modo, a medida que las filas migradas llegan al nodo maestro de la partición de destino, se envían a las réplicas de la partición de destino mediante el uso de la bitácora de registros. Por lo tanto, las réplicas de origen y destino se mantienen actualizadas a medida que avanza la migración.

- Una vez que se ha migrado una partición completa, el maestro de particiones de origen y el maestro de particiones de destino actualizan su versión del mapa de particiones. Tan pronto como el controlador de cliente tenga un mapa de partición actualizado, puede comenzar a enrutar las solicitudes de la aplicación al fragmento apropiado.
- Si el administrador del sistema desea reducir el número de particiones en el sistema, el administrador indica que es necesario quitar parte del hardware del clúster.
  - Esto inicia una operación de transferencia de particiones que aumenta el número de particiones por partición, reduciendo en consecuencia el número total de particiones en el sistema.
  - Una vez que se completa la transferencia, las máquinas que ya no administran los datos se pueden eliminar físicamente del sistema.

- Las claves en Oracle NoSQL Database se dividen en dos partes discretas:
  - *Componente mayor.* El componente mayor de una clave denota el fragmento que contendrá los registros de todas las claves menores que se derivan de una clave mayor específica.
    - Esto significa que se garantiza que los registros que comparten la misma combinación de componentes clave mayores estén en el mismo fragmento, y se pueden consultar de manera eficiente.
    - Además, los registros con componentes clave mayores idénticos pueden participar en las transacciones ACID.
    - Las claves se distribuyen por todo el almacén "hasheando" en el componente mayor de la clave.
  - *Componente menor*. El componente de clave menor de una clave se puede considerar como la ruta local de fragmentación al registro.
    - "Hasheando" el componente mayor indicará el fragmento que contiene los datos; el uso del componente menor de la clave indicará el registro en esa partición.
- Toda la clave, componentes mayor y menor, deben ser únicos.

Oracle NoSQL Database utiliza varias técnicas de compresión de claves (por ejemplo, compresión de prefijos) para el índice del árbol B. Esto garantiza una utilización óptima de la memoria y una recuperación eficiente de uno o más pares clave-valor de un registro específico.

Admite la noción de iteración ordenada en las claves menores dentro de un registro (clave principal única).

Admite consultas de rango, lo que permite a la aplicación recuperar solo un subconjunto de los datos.

También puede almacenar información como registros JSON utilizando esquemas Avro de la siguiente manera:

```
{
     "type" : "record",
     "name" : "userInfo",
     "namespace" : "my.example",
     "fields" : [{"First_name" :
"firstname",
"type" : "string","default" : "NONE"},
                {"Last_name" : "lastname",
"type" : "string", "default" : "NONE"},
                {"name" : "address",
"type" : "record","fields": [
                              {"name" : "street",
"type" : "string","default" : "NONE"},
                              {"name" : "city",
"type" : "string", "default" : "NONE"},
                              {"name" : "state",
"type" : "string", "default" : "NONE"},
                              {"name" : "zipcode",
"type" : "string", "default" : "NONE"}]
]
}
```

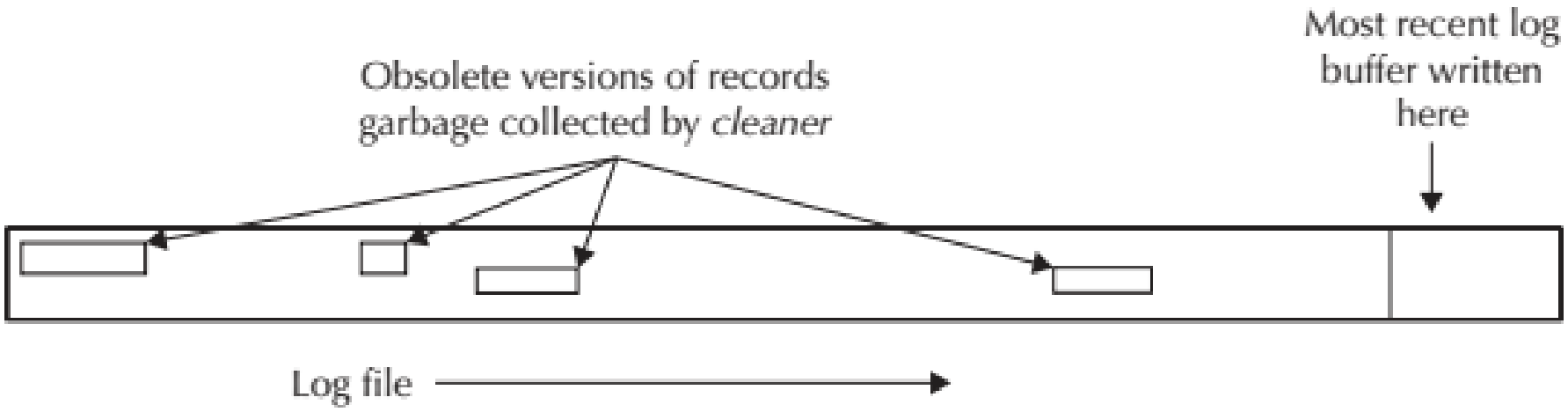
# Modelo de datos en Oracle NoSQL

- El paradigma clave-valor de la base de datos se puede utilizar fácilmente para modelar estructuras de datos relacionales.
  - También se puede utilizar para administrar objetos Java, estructuras de datos jerárquicas, estructuras de datos anidadas, datos JSON, datos XML, tablas dispersas, etc.
  - Oracle NoSQL Database también se puede utilizar como repositorio de datos de gráficos RDF.
- Esta flexibilidad de modelado simplifica la tarea del desarrollador de aplicaciones porque le permite modelar los datos de la manera más conveniente para la aplicación.
- Además, es muy fácil para el desarrollador de aplicaciones evolucionar el modelo de datos; los cambios se pueden incorporar fácilmente sin tener que descargar y volver a cargar los datos existentes.

- La arquitectura estructurada por registros es una arquitectura de solo inserción.
- La base de datos está organizada como un único archivo de bitácora lógico en el disco. El archivo de bitácora contiene todos los datos, incluidos los metadatos, como los índices.
  - Cuando se inserta un nuevo registro, se anexa al final de la bitácora. Si se cambia un registro existente, se crea una nueva versión del registro y se anexa al final de la bitácora en lugar de cambiar la copia existente del registro.
  - Por lo tanto, cada cambio da como resultado que se cree una nueva versión del registro en lugar de que el registro se actualice en su lugar.
  - Por supuesto, los índices y otras estructuras relacionadas deben actualizarse para reflejar la nueva posición de la nueva versión del registro en disco.
  - Cuando se actualiza un registro, la versión anterior del registro ya no es necesaria y se puede recopilar como elemento no utilizado.

# Almacenamiento estructurado en registros

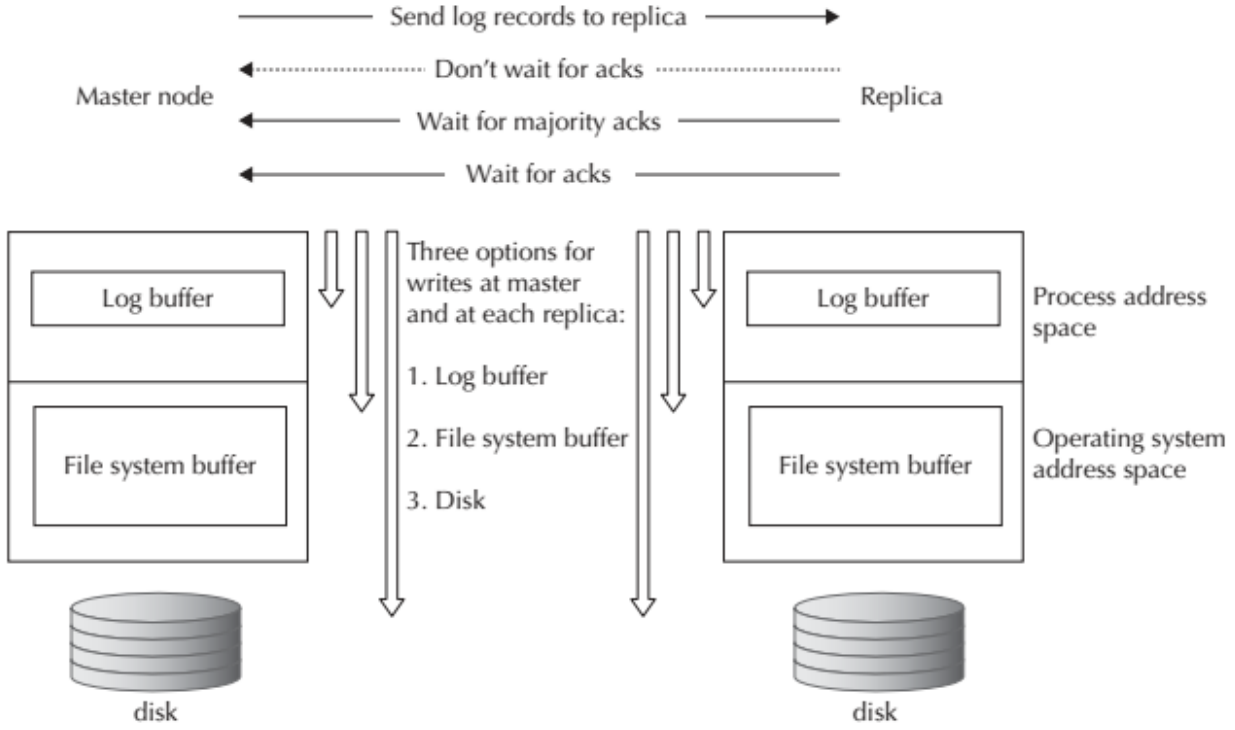
- Se incluye un proceso de recolección de basura llamado limpiador (cleaner).
  - El limpiador es responsable de recuperar el espacio asociado con versiones obsoletas de los registros.
  - El limpiador se ejecuta en segundo plano, buscando registros obsoletos en el archivo de bitácora, y compacta el archivo de bitácora moviendo los registros actuales al final y recuperando espacio en el disco. Aunque la limpieza de registros es una actividad en segundo plano, es importante optimizar el funcionamiento del proceso de limpieza para que no interfiera con el rendimiento general del sistema.



# Durabilidad

- El protocolo de escritura anticipada (WAL) es una técnica de optimización que requiere que solo los registros de bitácora asociados con los cambios se escriban en el disco antes de que se pueda declarar completada la transacción.
- El protocolo de confirmación de grupo está diseñado para escribir varios registros de bitácora como un grupo en una sola E/S en el archivo de bitácora.
- Es importante hacer que los cambios sean duraderos en el nodo maestro, así como propagar los cambios a las réplicas antes de que la transacción se pueda declarar como completa.
- Cuando una transacción está lista para ser confirmada, Oracle NoSQL Database hace lo siguiente (en paralelo):
  - Escribir los cambios de transacción en el archivo de bitácora en el nodo maestro
  - Enviar mensajes que contengan los registros de bitácora a las réplicas

- En el contexto de la escritura de los registros de bitácora en el disco, hay tres posibilidades a considerar:
  - Escribir el registro de bitácora en el búfer de bitácora.
  - Escribir el búfer de bitácora en el sistema de archivos.
  - Escribir el búfer de bitácora en el disco (llamada **fsync()** del sistema de archivos).
- Hay que tener en cuenta que estas opciones para escribir registros de bitácora en el disco están disponibles en el nodo maestro, así como en las réplicas.
- Oracle NoSQL Database permite al usuario especificar el grado de durabilidad para los nodos maestros y réplicas.



# Instalación de Oracle NoSQL Database

- El sistema operativo debe configurarse con el conjunto apropiado de paquetes que requiere el software Oracle NoSQL Database. Por lo tanto, debe asegurarse de que todos los nodos de almacenamiento del almacén clave-valor cumplen los siguientes requisitos:
  - El sistema operativo Oracle Linux y Oracle Solaris son los sistemas operativos oficialmente soportados para Oracle NoSQL Database. Es importante asegurarse de que el SO subyacente sea totalmente compatible con Oracle.
  - Java. Asegúrese de que Java SE 6 (JDK 1.6.0 u25) o posterior está instalado en los nodos de almacenamiento. De lo contrario, instale la versión correcta de Java desde el sitio de descarga de Oracle (*http://www.oracle.com/technetwork/java/javase/downloads/.html*). Puede utilizar el comando *java –version* para comprobar la versión de Java instalada en el sistema.
  - El sistema de archivos para *KVHOME* y *KVROOT. KVHOME* es la ubicación del sistema de archivos que almacena los binarios de software de Oracle NoSQL Database y *KVROOT* es la ubicación para almacenar los archivos de configuración de Oracle NoSQL Database y también sirve como la ubicación predeterminada para los datos de pares clave-valor.
  - Puertos de red. Los nodos de almacenamiento y los nodos de replicación del almacén clave-valor se comunican entre sí mediante el protocolo TCP/IP a través de puertos.

- Una vez que haya descargado el software, copie el archivo Zip en todos los nodos de almacenamiento y muévalos al directorio raíz de *KVHOME*. Extraiga el contenido del paquete utilizando las utilidades de descomprimir apropiadas (*gunzip* seguido de *tar* para **.gz* y descomprimir para **.zip*), y cuando se complete la extracción, el directorio *KVHOME* se crea automáticamente.
- Repita este proceso en todos los nodos de almacenamiento y, una vez que la extracción se realiza correctamente en todos los nodos, habrá completado la instalación del software Oracle NoSQL Database.

```
$ cd /u01/kvhomes
$  gunzip  kv-ee-2.0.39.tar.gz
$  tar  xvf  kv-ee-2.0.39.tar
$  ls  -F /u01/kvhomes/kv-2.0.39
build.xml  doc/  examples/  exttab/  lib/LICENSE.txt README.txt
```

- Se puede verificar la instalación ejecutando una aplicación de prueba suministrada llamada *kvclient.jar*

```
$ java -jar  /u01/kvhomes/kv-2.0.39/lib/kvclient.jar
11gR2.2.0.39
```

- Opciones del comando *kvclient.jar kvlite*:

```
[-root <rootDirectory>]          # defaults to: ./kvroot
[-store <storeName>]             # defaults to: kvstore
[-host <hostname>]               # defaults to: local host name
[-port <port>]                   # defaults to: 5000
[-noadmin]                       # defaults to: false
[-secure-config <enable|disable>]# defaults to: enable
```

- Se recomienda crear un directorio particular para el almacenamiento de los datos y establecerlo en la opción `-root`. De igual manera, establecer el nombre del store en la opción `-store`, y establecer la opción de seguridad como `disable` para sistemas de desarollo y prueba.

# Servicio de administración de bases de datos Oracle NoSQL

- Oracle NoSQL Database Administration Service es un proceso que se ejecuta en los nodos de almacenamiento y está a cargo de una variedad de actividades de administración en Oracle NoSQL Database, como el inicio/apagado de instancias, la configuración inicial, las modificaciones continuas en la configuración y la supervisión del rendimiento del sistema.
- El Servicio de Administración también se encarga de recopilar y mantener estadísticas de rendimiento de la base de datos, así como de registrar eventos importantes del sistema y, por lo tanto, de ayudar a supervisar en línea y ayudar a ajustar el rendimiento de la base de datos.
- La práctica recomendada es tener un mínimo de tres servicios de administración para que se tenga que al menos un servicio disponible en un tiempo determinado.
- Se puede acceder al Servicio de administración desde una interfaz de línea de comandos denominada CLI y una consola basada en web denominada Consola de Administración Web.

# Interfaz de línea de comandos (CLI) de administración

- La CLI de administración admite todas las actividades de administración en el almacén clave-valor.
- Para iniciar la CLI de administración, ejecute la clase `runadmin` de `kvstore.jar`. El `runadmin` interactúa con el servicio de administración y proporciona el prompt de la CLI (`kv->`)
- Debe asegurarse de que el servicio Agente de Nodo de Almacenamiento se inicie antes de ejecutar `runadmin`
- La CLI se puede invocar principalmente en tres modos: un modo interactivo, un modo de comando único y un modo de script.
  - El modo interactivo es el más utilizado y es el único modo que proporciona un símbolo del sistema (el símbolo del sistema `kv->`).

- El modo de comando único, *single command mode*, por otro lado, ejecuta un solo comando directamente en el símbolo del sistema operativo mientras invoca la CLI. Se pasa el comando CLI como parámetro a `runadmin`.
- Una vez que el comando CLI completa su ejecución, el control se devuelve al sistema operativo.
- Si el comando se completa correctamente, el código de salida devuelto al sistema operativo es 0 y, si encuentra un error, el código de salida es un valor distinto de 0.
- El uso general de invocar la CLI en el modo de comando único es el siguiente:

```
java -jar KVHOME/lib/kvstore.jar runadmin
-host <hostname>-port <port>[single command and arguments]
```

- donde `<hostname>` y `<port>` son el nombre de host SNA y el puerto del Registro, respectivamente.
- A continuación se muestra un ejemplo de cómo iniciar la CLI mediante `runadmin` para acceder al SNA en *node01* con el puerto del Registro 5000:

```
java -jar KVHOME/lib/kvstore.jar runadmin -host node01 -port 5000
kv->
```

# Opciones de kvstore.jar

```
Usage: java -jar KVHOME/lib/kvstore.jar
  <kvlite |
   makebootconfig |
   securityconfig |
   diagnostics |
   start |
   stop |
   status |
   restart |
   runadmin |
   load |
   ping |
   version |
   generateconfig |
   help> [-verbose] [args]
Use "help <commandName>" to get usage for a specific
command
Use "help commands" to get detailed usage information
Use the -verbose flag to get debugging output
```

- El modo de script es muy similar al modo de comando único, pero ejecuta un script que contiene varios comandos de CLI en lugar de ejecutar solo un solo comando de CLI. Con el modo script, es fácil automatizar tareas repetitivas o ejecutar tareas en modo por lotes, sin requerir la supervisión directa de un administrador.
- Al igual que con el modo de comando único, el control vuelve al símbolo del sistema operativo una vez que se completa el script. El archivo de script se especifica mediante el modificador load `-file` al invocar `runadmin`. El siguiente es un ejemplo típico de CLI en modo script:

```
java -jar  KVHOME/lib/kvstore.jar runadmin -host <hostname> -port <port>
```

- Puede utilizar el comando `help` para detectar todos los comandos permitidos por la CLI o anexar un indicador `-help` a un comando específico para mostrar su sintaxis de uso.

```
Usage: java -jar KVHOME/lib/kvstore.jar runadmin
        -host <hostname> -port <port> [-store <storeName>]
        [-admin-host <adminHost> -admin-port <adminPort>]
        [-username <user>] [-security <security-file-path>]
        [-admin-username <adminUser>] [-admin-security <admin-security-file-path>]
        [-timeout <timeout ms>] [-consistency <NONE_REQUIRED(default) | ABSOLUTE |
NONE_REQUIRED_NO_MASTER>]
        [-durability <COMMIT_SYNC(default) | COMMIT_NO_SYNC |
COMMIT_WRITE_NO_SYNC>]
        [-dns-cachettl <time in sec>]
        [single command and arguments]
```

- Además de la CLI de administración, la Consola de Administración Web también se puede utilizar para administrar Oracle NoSQL Database.
- La Consola de Administración Web admite principalmente actividades de administración de tipo de solo lectura, como examinar la topología del store, supervisar las ejecuciones del plan y examinar el archivo de registro de todo el clúster; sin embargo, no admite actividades relacionadas con la configuración o modificación del store. Estas son tareas manejadas por la CLI de administración.
- Para acceder a la Consola de Administración Web, acceda al host que ejecuta el Servicio de Administración y especifique el Puerto de Administración.
  - Por ejemplo, usaría la dirección URL *http://localhost:5001* para acceder a la Consola de Administración Web con el Servicio de Administración ejecutándose en *localhost* y escuchando en el puerto 5001.

# Consola de administración web

- Hasta ahora, solo tiene el software Oracle NoSQL Database instalado en los nodos de almacenamiento. El siguiente paso consiste en realizar algunas tareas adicionales, como especificar los puertos de red para los Agentes de Nodo de Almacenamiento (SNA), la Consola de Administración Web y los Puertos de Replicación, y la ubicación *KVROOT* para almacenar los archivos de configuración y, opcionalmente, los archivos de datos.
- El proceso del SNA y el Servicio de Administración utilizan un archivo de configuración en el inicio para configurar los puertos de red y otros parámetros de inicialización. Este archivo de configuración también se conoce como el archivo de configuración de arranque y se encuentra en el directorio *KVROOT* con un nombre predeterminado de *config.xml*.
- Para un nodo de almacenamiento recién instalado, el archivo de configuración de arranque no existe y debe crearse manualmente con la utilidad `makebootconfig`. La utilidad `makebootconfig` tiene la sintaxis siguiente:

# Crear la configuración de arranque

```
Usage: java -jar KVHOME/lib/kvstore.jar makebootconfig
        -root <rootDirectory> -host <hostname> -harange <startPort,endPort>
        -port <port>
        [-store-security <configure|enable|none>]
        [-noadmin]
        [-force][-dns-cachettl <time in sec>][-config <configFile>]
        [-storagedir <directory path>][-storagedirsize <directory size>][-capacity <n_rep_nodes>]
        [-num_cpus <ncpus>][-memory_mb <memory_mb>]
        [-servicerange <startPort,endPort>]
        [-hahost <haHostname>]
        [-secdir <security dir>] [-pwdmgr {pwdfile | wallet | <class-name>}]
        [-kspwd <password>]
        [-external-auth {kerberos}]
          [-krb-conf <kerberos configuration>]
          [-kadmin-path <kadmin utility path>]
          [-instance-name <database instance name>]
          [-admin-principal <kerberos admin principal name>]
          [-kadmin-keytab <keytab file>]
          [-kadmin-ccache <credential cache file>]
          [-princ-conf-param <param=value>]*
        [-security-param <param=value>]*
        [-mgmt {jmx|none}]
```

- Los siguientes son los detalles sobre los parámetros comúnmente utilizados de `makebootconfig`.
  - `-root <rootDirectory>` Es la ubicación *KVROOT* que almacena los archivos de configuración y, opcionalmente, los datos del par clave-valor.
  - `-port <port>` Cada el Nodo de Almacenamiento ejecuta un proceso de SNA para facilitar las comunicaciones entre otros procesos de SNA y las aplicaciones cliente. El SNA escucha en el puerto del Registro especificado mediante el parámetro `-port`. El puerto de registro que se utiliza normalmente es 5000.
  - `-harange <startPort, endPort>` Cada nodo de almacenamiento requiere un conjunto de puertos (especificados mediante un rango, denominados puertos de Rango de Alta Disponibilidad HA) que utilizarán los nodos de replicación y los servicios de administración para facilitar la replicación de datos de usuario (pares clave-valor). Los puertos se especifican como un rango usando "`startPort, endPort`".

- Continúe con la creación del archivo de configuración de arranque, como se muestra en el siguiente ejemplo. Se llama al comando `makebootconfig` con */u02/kvroot* como *KVROOT*, al SNA que se ejecuta en el puerto 5000, al servicio de administración que se ejecuta en el puerto 5001, al intervalo de puertos de intervalo como 5010–5020.

```
$> mkdir -p /u02/kvroot
$> java -jar /u01/kvhomes/kv-2.0.39/lib/kvstore.jar makebootconfig
-root /u02/kvroot -host localhost -port 5000 -harange  5010,5020
```

- El siguiente paso es iniciar los procesos del SNA en cada uno de los nodos de Oracle NoSQL Database. El SNA inicia automáticamente el Servicio de Administración de Bootstrap.
- Puede utilizar la utilidad `start` para iniciar procesos de SNA, y también recuerde iniciar el SNA en todos los nodos de almacenamiento que se utilizarán para configurar la base de datos Oracle NoSQL.
- En el ejemplo siguiente se inicia el proceso SNA utilizando `/u02/kvroot` como *KVROOT*:

```
$> nohup java -jar /u02/kvroot/lib/kvstore.jar start -
root /u02/kvroot &
```

- Una vez completados todos los pasos anteriores, se configura un conjunto de nodos de almacenamiento para que actúe y funcione como un clúster distribuido del store.
- El proceso de configuración del store comprende los siguientes pasos:
  1. Inicie la CLI de administración.
  2. Asigne un nombre al store.
  3. Cree un centro de datos.
  4. Implemente el primer nodo de almacenamiento.
  5. Crear un servicio de administración.
  6. Cree un grupo de nodos de almacenamiento.
  7. Cree los nodos de almacenamiento restantes.
  8. Crear e implementar nodos de replicación.

# Pasos de configuración

# Iniciar la CLI de administración

- Los pasos de configuración se realizan mediante la CLI de administración. Antes de invocar la CLI, seleccione el nodo de almacenamiento que serviría como nodo de administración principal durante el proceso de configuración y también el nodo que contiene la copia maestra de la base de datos de administración.
- Inicie sesión en el nodo de almacenamiento que ha identificado como el nodo de administración principal. Inicie la CLI invocando `runadmin`, como se muestra en el ejemplo siguiente:

```
> java –jar KVHOME/lib/kvstore.jar
runadmin -port 5000 -host localhost
kv->
```

# Asignar un nombre al store

- Uno de los primeros atributos que configurará es el nombre del store
- El nombre que elija debe ser adecuado para la función, la aplicación y/o el contenido del store.
- El nombre del store se utiliza para crear una ruta de directorio en el sistema de archivos, bajo la cual se crearán subdirectorios para almacenar los pares clave-valor reales. Por lo tanto, sintácticamente hablando, cualquier nombre sería válido siempre y cuando lo permita el sistema operativo para nombrar directorios.
  - Los caracteres válidos admitidos para un nombre de store son caracteres alfanuméricos, un signo menos (–), un guión bajo (_) y un punto (.).
- En el símbolo del sistema `kv->`, utilice el comando `-name` para nombrar el store. Este comando toma el nombre del store como parámetro (el único parámetro) y garantiza que el nombre sea sintácticamente válido y permitido por el sistema. De lo contrario, se marca un error. En el ejemplo que se muestra aquí, a un store se le da el nombre *movieDBstore*:

```
kv-> configure -name movieDBstore
```

# Implementar el primer nodo de almacenamiento

- Se debe agregar el primer nodo de almacenamiento al store. Aunque ya se ha conectado a un nodo de almacenamiento e iniciado el servicio SNA, no se ha agregado al store.
- Ejecute el comando `plan deploy-sn` para agregar el nodo de almacenamiento al store. Este comando toma el ID de la zona de datos como entrada, que se obtiene mediante el comando `show topology`.

```
kv-> show topology
store=empresa  numPartitions=10 sequence=14
  zn: id=zn1 name=KVLite repFactor=1 type=PRIMARY allowArbiters=false
  sn=[sn1] zn:[id=zn1 name=KVLite] localhost:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
             single-op avg latency=1.4431906 ms   multi-op avg latency=0.0 ms
  shard=[rg1] num partitions=10
    [rg1-rn1] sn=sn1
```

- Ahora que tiene el ID del centro de datos, ejecute plan `deploy-sn` y agregue el nodo de almacenamiento 01 con el puerto del Registro de 5000 al ID de la zona de datos *zn1*, como se muestra en el ejemplo siguiente:

```
kv-> plan deploy-sn -zn zn1 -host node01 -port 5000 -wait
Executed plan 2, waiting for completion...
Plan 2 ended successfully
```

- Opciones:

```
Usage: plan deploy-sn -zn <id> | -znname <name> -host <host> -port <port>
       [-json]
       [-plan-name <name>] [-wait] [-noexecute] [-force]
       Deploys the storage node at the specified host and port into the
       specified zone.
```

# Crear el servicio de administración

- El Servicio de Administración se encarga de mantener la base de datos de administración y de proporcionar una Consola de Administración basada en Web. El paso después de crear el primer nodo de almacenamiento es crear el Servicio de Administración mediante el comando `plan deploy-admin`. Este comando requiere el ID de Nodo de Almacenamiento (obtenido del comando de topología) y el número de puerto HTTP del Servicio de Administración.
- El número de puerto de administración se utiliza para enrutar el tráfico HTTP a la Consola de Administración Web. En el ejemplo siguiente se implementa el Servicio de Administración en el identificador de nodo de almacenamiento *sn1* con el puerto de administración 5001.

```
kv-> show topology
store=empresa  numPartitions=10 sequence=14
  zn: id=zn1 name=KVLite repFactor=1 type=PRIMARY
allowArbiters=false
  sn=[sn1] zn:[id=zn1 name=KVLite] localhost:5000
capacity=1 RUNNING
    [rg1-rn1] RUNNING
            single-op avg latency=1.4431906 ms   multi-
op avg latency=0.0 ms
  shard=[rg1] num partitions=10
    [rg1-rn1] sn=sn1


kv-> plan deploy-admin -sn sn1 -wait
Executed plan 3, waiting for completion...
Plan 3 ended successfully
```

- Un Grupo de Nodos de Almacenamiento es una agrupación lógica de todos los nodos de almacenamiento que están presentes en el store. Los Nodos de Almacenamiento se asocian con grupos para facilitar la distribución óptima de los recursos, especialmente cuando los nodos de almacenamiento se agregan o eliminan del store.
- Una vez que haya creado el Servicio de Administración, cree un Grupo de Nodos de Almacenamiento mediante el comando `pool create`. El comando requiere el nombre del grupo como entrada y se ejecuta solo una vez en el momento de la creación del grupo. A continuación, agregue los nodos de almacenamiento al grupo mediante el comando `pool join`. Debe ejecutar el comando `pool join` en todos los nodos de almacenamiento, incluido el creado anteriormente. El comando `pool join` asocia un nodo de almacenamiento al grupo y requiere el nombre del grupo y el identificador del nodo de almacenamiento como entrada.

```
Usage: pool clone |
            create |
            join |
            leave |
            remove
```

- En el ejemplo siguiente se muestran los comandos de la CLI que se ejecutarán en este paso:

```
kv-> pool  create  -name  movieDBpool

kv  -> pool  join  -name  movieDBpool  -sn sn1
Added  Storage  Node(s) [sn1]  to  pool  movieDBpool
```

# Crear un Grupo de Nodos de Almacenamiento

- Actualmente se tiene un store de un solo nodo (con un factor de replicación de uno), que no proporciona la alta disponibilidad que normalmente se requiere para las implementaciones de producción. Por lo tanto, debe implementar nodos de almacenamiento adicionales en el store y agregarlos al grupo de nodos de almacenamiento.
- Utilice el comando plan `deploy-sn` para implementar el nodo de almacenamiento en el clúster y el comando `pool join` para agregar el nodo de almacenamiento al grupo de nodos de almacenamiento. Recuerde repetir estos comandos en todos los nodos de almacenamiento que haya identificado como parte del store.
- Se ilustra la adición del nodo de almacenamiento 02 y 03 al store.

```
kv-> plan  deploy-sn  -dc  dc1 -host node02 -port  5000 -wait
kv-> pool  join  -name  movieDBpool  -sn  sn2
kv-> plan  deploy-sn  -dc  dc1 -host node03 -port  5000 - wait
kv-> pool  join  -name  movieDBpool  -sn  sn3
....
```

# Crear los nodos de almacenamiento restantes

- Por ejemplo, si el nodo de almacenamiento que se acaba de crear tiene un ID de 5, al siguiente nodo de almacenamiento que cree se le asignará un ID de 6.
- Por lo tanto, siempre puede predecir el ID de nodo de almacenamiento del nodo que acaba de implementar incrementando el ID del nodo de almacenamiento anterior en uno, lo que significa que puede ejecutar directamente `deploy-sn` sin la necesidad de ejecutar `show topology`.
- Tenga en cuenta que esto solo es cierto siempre que solo se use una sesión de CLI para ejecutar los comandos `deploy-sn`.

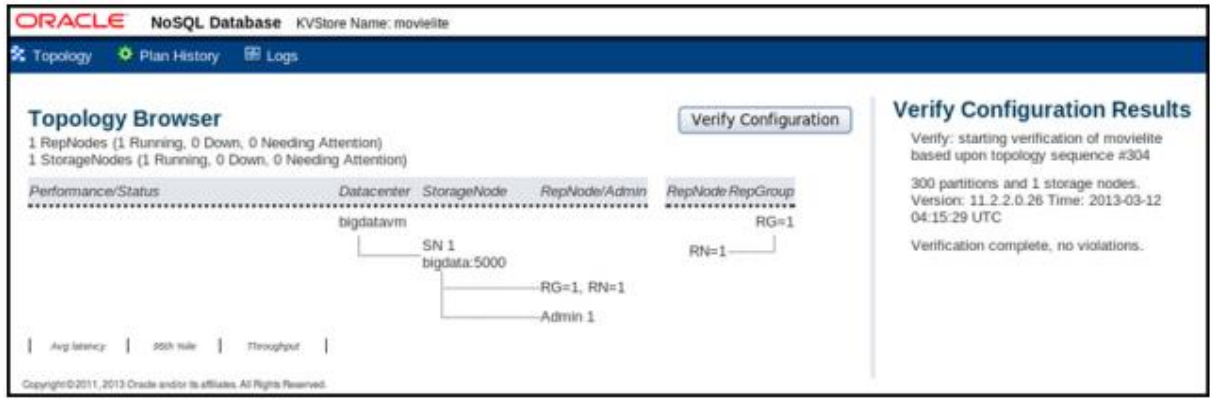
- El comando `create topology` se utiliza para crear la topología y requiere el nombre de la topología, el grupo de nodos de almacenamiento y el número total de particiones como entrada. El nombre de la topología es un nombre único que se define para identificar la topología; el grupo de nodos de almacenamiento es el nombre del grupo de nodos de almacenamiento que creó anteriormente; y el número total de particiones se obtiene mediante un ejercicio de planificación y dimensionamiento de la capacidad
- El comando de creación de topología creará automáticamente un número adecuado de particiones y nodos de replicación en función del número de nodos de almacenamiento y el factor de replicación.
- En el ejemplo siguiente se ilustra el uso de los comandos de creación e implementación de topología:

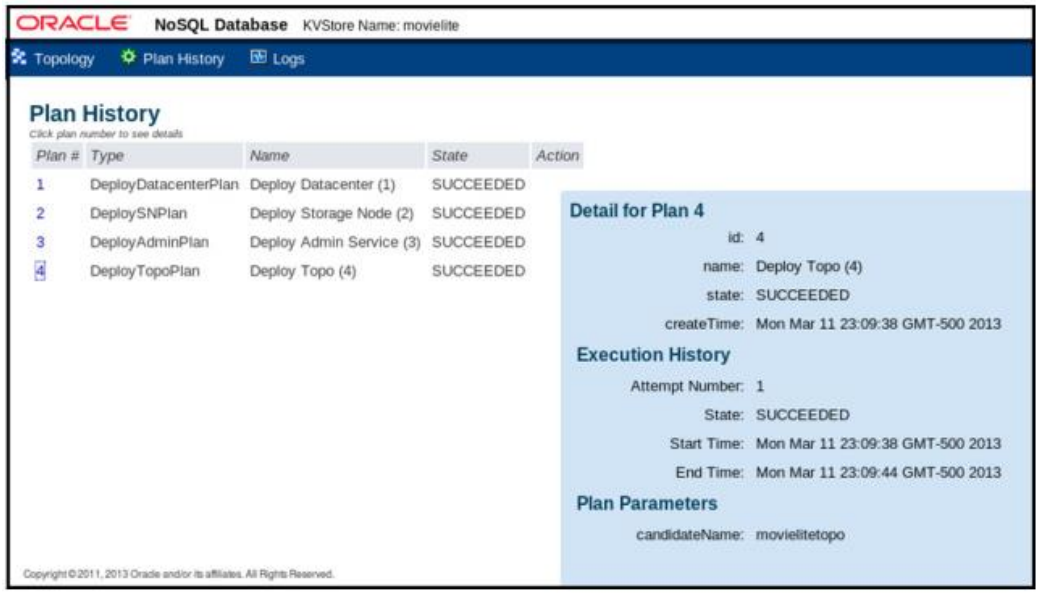
```
kv-> topology create -name movietopo -pool movieDBpool -partitions 300
kv-> plan deploy-topology -name movietopo -wait
Executed plan 6, waiting for completion...
Plan 6 ended successfully
```

# Crear e implementar nodos de replicación

- El store se instala y configura completamente una vez que los comandos anteriores se completan correctamente.
- La verificación se puede hacer mediante la Consola de Administración Web y la CLI

- La Consola de Administración Web y la CLI se pueden usar para validar la topología del store y observar los resultados de la ejecución del plan.
  - Si utiliza la Consola de Administración Web, use el navegador Web con el equipo (hostname) y el puerto que ejecutan el Servicio de Administración. Por ejemplo, *http://node01:5001* para iniciar la Consola de Administración Web.






- Hay que asegurarse de que la información del centro de datos, el nodo de almacenamiento, el nodo de replicación y el nodo de administración se muestre correctamente y que los procesos tengan el estado *RUNNING*.
- Alternativamente, también puede usar la CLI de administración para verificar la configuración. En el símbolo del sistema de la CLI, ejecute los comandos **`show plans`** y **`show topology`**, como se muestra en el ejemplo siguiente. Los comandos deben mostrar el estado de *SUCCEEDED* para las ejecuciones del plan y el estado de *RUNNING* para los SNA y los Servicios de Administración (Administration Services).

```
kv-> show plans
1 Deploy Datacenter (1) SUCCEEDED
2 Deploy Storage Node (2) SUCCEEDED
3 Deploy Admin Service (3) SUCCEEDED
4 Deploy Topo (4) SUCCEEDED
kv-> show topology
store=empresa  numPartitions=10 sequence=14

  zn: id=zn1 name=KVLite repFactor=1 type=PRIMARY
allowArbiters=false

  sn=[sn1] zn:[id=zn1 name=KVLite] localhost:5000
capacity=1 RUNNING

    [rg1-rn1] RUNNING

              single-op avg latency=1.0870148 ms
multi-op avg latency=0.0 ms

  shard=[rg1] num partitions=10

    [rg1-rn1] sn=sn1
```

# Utilidades

- El servidor *kvstore.jar* cuenta con un intérprete de comando para administración de tablas y datos, que se ejecuta con el comando `java -jar kvstore.jar runadmin -host localhost -port 5000` Las opciones disponibles son las siguientes:

```
Oracle NoSQL Database Administrative Commands:
        aggregate
        await-consistent
        change-policy
        configure
        connect
        ddl
        delete
        execute
        exit
        get
        help
        history
        load
        logtail
        page
        ping
        plan
        pool
        put
        repair-admin-quorum
        show
        snapshot
        table-size
        timer
        topology
        verbose
        verify
```

- El commando *show* permite obtener un listado de las tablas, la topología y otros datos de configuración:

```
kv->show tables
kv->show topology
kv->show schemas
kv->show schema -name <table>
```

- El comando *ddl* permite agregar un esquema al store actual

```
kv->ddl add-schema -file ../persona.avsc
```

- El esquema deberá seguir la especificación de almacenamiento basada en Avro.

# Desarrollo en KVLite

- *KVLite* es una versión ligera del servidor de base de datos NoSQL que se ejecuta en un solo nodo, tiene un solo grupo de replicación y está empaquetado dentro del archivo *kvstore.jar* ubicado en la carpeta `lib` del directorio `KVHOME`.
- *KVLite* se puede iniciar utilizando el siguiente comando:

```
java –jar KVHOME/lib/kvstore.jar
kvlite -root <./kvroot> -store
<nombre kvstore> -host localhost
–port 5000
```

- Cuando se inicie *KVLite* correctamente, se creará un nuevo store o se abrirá un store existente si se creó antes. Los diferentes parámetros para usar con la utilidad de línea de comandos *KVLite* son los siguientes:

  `-logging` Activa el registro de aplicaciones Java. Los archivos de registro se colocan en el directorio de ejemplos de la distribución de Oracle NoSQL Database.

  `-root` Identifica la ruta del directorio principal de Oracle NoSQL Database. En el caso de *KVLite*, los archivos de base de datos del store se encuentran en local. El directorio tiene que estar presente, y si los archivos de base de datos no están presentes, se crearán.

  `-store` Identifica el nombre del store. Esta opción solo debe usarse si está creando un nuevo store.

  `-host` Esta opción especifica el nombre del host en el que se ejecuta KVLite. El nombre de host DNS debe usarse si se desea conectarse a la instancia KVLite desde otro equipo.

  `-port` Identifica el puerto en el que KVLite está escuchando las conexiones de cliente.

```
{
"type":"record",
"name":"Persona",
"namespace":"test",
"fields":[
        {"name":"nombre", "type":{
                "type":"record",
                "name":"NombreCompleto",
                "fields":[{"name":"nombre", "type":"string", "default":""},
                        {"name":"apellido", "type":"string", "default":""}]
                }, "default":{}
        },
        {"name":"edad", "type":"int", "default":0},
        {"name":"direccion", "type":{
                "type":"record", "name":"Direccion", "fields":[
                        {"name":"calle", "type":"string", "default":""},
                        {"name":"ciudad", "type":"string", "default":""},
                        {"name":"estado", "type":"string", "default":""},
                        {"name":"cp", "type":"int", "default":0}
                        ]
                }, "default":{}
        }
        ]
}
```

Para crear una tabla y su estructura de almacenamiento se puede emplear el comando `table` y la definición de los registros con el comando `add-field`:

```
Usage: table clear |
               create |
               evolve |
               list

Usage: add-field -type <type> -name <name> [-not-nullable]
         [-default <value>][-desc <description>]
         [-size <size>] [-enum-values <valuxite[,value[,...]]>]
         <type>: INTEGER, LONG, DOUBLE, FLOAT, STRING, BOOLEAN,
BINARY, FIXED_BINARY, ENUM.
         Add a field. Ranges are inclusive with the exception of
         String, which will be set to exclusive.
```

- Para estructuras es posible establecer varios nivel, que corresponden a las claves mayores y menores, mediante las siguientes construcciones:

```
Usage: add-record-field -name <name> [-desc <description>]
        Build a record field.
Usage: add-array-field -name <name> [-desc <description>]
        Build a array field.
Usage: add-map-field -name <name> [-desc <description>]
        Build a map field.
Usage: primary-key -field <name> [-field <name>]+
        Set primary key.
Usage: remove-field -name <name>
        Remove a field.
Usage: set-description -desc <description>
        Set description for the table.
Usage: shard-key -field <name> [-field <name>]+
        Set shard key.
Usage: add-schema -name <schema-name>
        Build a table from Avro schema.
```

# Ejemplo de creación de una tabla

```
kv->table create -name usuario
usuario->add-field -name Id -type integer
usuario->add-field -name nombre -type string
usuario->add-field -name apellidos -type string
usuario->add-record-field -name domicilio
usuario.domicilio->add-field -name calle -type string
usuario.domicilio->add-field -name numero -type integer
usuario.domicilio->add-field -name colonia -type string
usuario.domicilio->add-field -name ciudad -type string
usuario.domicilio->exit
usuario->add-field -name genero -type ENUM -enum-values M,F
usuario->add-field -name edad -type integer
usuario->primary-key -field Id
usuario->show
usuario->exit
```

- El comando `plan` permite establecer algunas opciones administrativas para las tablas creadas, así como el despliegue de las estructuras en los nodos de datos y las réplicas:

```
Usage: plan add-index |
             add-table |
             cancel |
             change-parameters |
             change-storagedir |
             deploy-admin |
             deploy-sn |
             deploy-topology |
             deploy-zone |
             evolve-table |
             execute |
             interrupt |
             migrate-sn |
             remove-admin |
             remove-index |
             remove-sn |
             remove-table |
             remove-zone |
             repair-topology |
             start-service |
             stop-service |
             wait
kv->plan add-table -name usuario -wait
```

- Se agregan datos a la tabla con el comando `put`, con la siguiente sintaxis:

```
Usage: put kv -key <key> -value <valueString> [-file] [-hex | -json <schemaName>]
        [-if-absent | -if-present]
        Puts the specified key, value pair into the store
        -file indicates that the value parameter is a file that contains the
        actual value
        -hex indates that the value is a BinHex encoded byte value with Base64
        -json indicates that the value is a JSON string.
        -json and -file can be used together.
        -if-absent indicates to put a key/value pair only if no value for
        the given key is present.
        -if-present indicates to put a key/value pair only if a value for
        the given key is present.
kv->put kv -key /usuario/Id -value 1
kv->put kv -key /usuario/nombre -value 'Juan'
kv->put kv -key /usuario/apellidos -value 'Pérez López'
kv->put kv -key /usuario/genero -value M
kv->put kv -key /usuario/domicilio/calle -value 'Av. Juarez'
kv->put kv -key /usuario/domicilio/numero -value 256
kv->put kv -key /usuario/domicilio/colonia -value 'Independencia'
kv->put kv -key /usuario/domicilio/ciudad -value 'Cuernavaca'
```

- Para leer los datos, se emplea el comando `get` con las siguientes opciones:

```
Usage: get kv -key <key> [-json] [-file <output>] [-all] [-keyonly]
        [-valueonly] [-start <prefixString>] [-end <prefixString>]
        Performs a simple get operation on the key in the store.
        -key indicates the key (prefix) to use.  Optional with -all.
        -json should be specified if the record is JSON.
        -all is specified for iteration starting at the key, or with
        an empty key to iterate the entire store.
        -start and -end flags can be used for restricting the range
used
        for iteration.
        -keyonly works with -all and restricts information to keys.
        -valueonly works with -all and restricts information to values.
        -file is used to specify an output file, which is truncated.

kv->get kv -key /usuario -all
kv->get kv -key /usuario -all -keyonly
kv->get kv -key /usuario -all -valueonly
```

- La eliminación de un datos se realiza con el comando `delete`, mediante el identificador del registro:

```
Usage: delete kv [-key <key>] [-start
<prefixString>] [-end <prefixString>] [-all]
        Deletes one or more keys. If -all is
specified, deletes all keys starting at the
specified key. If no key is specified delete all
keys in the store.
        -start and -end flags can be used for
restricting the range used for deletion.
```

# Verificación mediante el programa de ejemplo y ping

- Es compilar y ejecutar la aplicación **Hello World** de ejemplo suministrada por la instalación del software Oracle NoSQL Database. La ejecución de la aplicación de ejemplo requiere varias bibliotecas de software de la instalación y garantiza que la conectividad con el store sea completamente funcional. En el ejemplo siguiente se muestran los pasos para ejecutar esta aplicación:

```
# cd to KVHOME
$> cd c:\KVHOME
# Compile the sample Hello World Application
$> javac -cp lib\kvclient.jar .\sistema\src\avro\HelloBigDataWorld.java
# Run the Application. Substitute the <hostname>, <port> and <kvstore>
with your settings
C:\KVHOME\sistema\build\classes>java -classpath .;c:\KVHOME\lib\kvclient.jar
avro.HelloBigDataWorld -port 5000 -host localhost -store empresa

Hello Big Data World!
```

- También puede ejecutar el comando ping de *kvstore.jar* desde el símbolo del sistema operativo:

```
$ java -jar kvstore.jar ping -port 5000 -host localhost
Pinging components of store empresa based upon topology sequence #14
10 partitions and 1 storage nodes
Time: 2021-12-02 19:31:27 UTC   Version: 12.1.4.3.11
Shard Status: healthy:1 writable-degraded:0 read-only:0 offline:0
Admin Status: healthy
Zone [name=KVLite id=zn1 type=PRIMARY allowArbiters=false]   RN Status: online:1
offline:0
Storage Node [sn1] on localhost:5000    Zone: [name=KVLite id=zn1 type=PRIMARY
allowArbiters=false]    Status: RUNNING   Ver: 12cR1.4.3.11 2017-02-17 06:52:09 UTC
Build id: 0e3ebe7568a0
        Admin [admin1]          Status: RUNNING,MASTER
        Rep Node [rg1-rn1]      Status: RUNNING,MASTER sequenceNumber:205 haPort:5006
```

# Un programa básico de Hello World

- El programa *HelloToNoSQLDB* es una pieza de código muy simple que escribe un solo par clave-valor y luego lee el valor asociado con la clave.
- Las API principales de Oracle NoSQL Database están escritas en Java, y cualquier programador que interactúe con estas API debe conocer el lenguaje de programación Java.

```
package helloNOSQL;
import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.Key;
import oracle.kv.Value;
import oracle.kv.ValueVersion;
public class HelloNOSQLWorld {
private final KVStore store;
public static void main(String args[]) {
    try {
        HelloNOSQLWorld example = new HelloNOSQLWorld(args);
        example.runExample();
        } catch (RuntimeException e) {
        e.printStackTrace();
    }
}
```

- El constructor de la clase espera tres argumentos, que son necesarios para abrir el store y escribir en él.
- Se tienen valores predeterminados para *storeName*, *hostName* y *hostPort*.
- Si el usuario no proporciona ningún parámetro de línea de comandos, se utilizan los parámetros predeterminados.
- Sin embargo, el usuario tiene la opción de pasar los valores para un store en particular ubicado en una máquina en particular, que se ha configurado con un puerto no estándar.

```
•  /**
* Analiza  args de la línea  de comandos y abre el store clave-valor.
*/
HelloBigDataWorld(String[]  argv){
String storeName = "kvstore";
String hostName = "localhost";
String hostPort = "5000";
final  int  nArgs  =  argv.length;
int  argc = 0;
while  (argc  <  nArgs){
   final  String  thisArg  =  argv[argc++];
   if  (thisArg.equals("-store")){
      if  (argc  <  nArgs){
         storeName  =  argv[argc++];
      }  else  {
         usage("-store requiere un argumento");
      }
   }  else  if  (thisArg.equals("-host")){
      if  (argc  <  nArgs){
         hostName  =  argv[argc++];
      }  else  {
      usage("-host requiere un argumento");
   }
  }  else  if  (thisArg.equals("-port")){
     if  (argc  <  nArgs){
       hostPort  =  argv[argc++];
     }  else  {
        usage("-port requiere un argumento");
     }

   }else  {
     usage("Argumento desconocido  :"  +  esto);
}
```

- Una vez que los argumentos se analizan correctamente, se crea una instancia de un objeto *KVStoreConfig* utilizando los parámetros de línea de comandos pasados o los valores predeterminados.
- A continuación, estos parámetros se utilizan para obtener un identificador para el store llamando al método `getStore` en la clase `KVStoreFactory`. Usando este identificador, tenemos acceso a todas las llamadas de la API de Oracle NoSQL Database para manipular datos.

```
store = KVStoreFactory.getStore(new KVStoreConfig(storeName, hostName +
":" + hostPort));
}
```

- La siguiente función solo imprime mensaje del uso correcto de parámetros.

```
private void usage(String message) {
 System.out.println("\n" + message + "\n");}
 System.out.println("uso: HelloBigDataWorld");
 System.out.println("\t-store <instance name> (default: kvstore) " +"-host
<host name> (default: localhost) " +"-port <port number> (default:
5000)");
 System.exit(1);
}
```

- Ahora se debe crear una instancia de un objeto `oracle.kv.Key` y una instancia de un objeto `oracle.kv.Value`. En general, las instancias de los objetos `oracle.kv.Key` se pueden crear a partir de *java Strings* y las instancias de los objetos `oracle.kv.Value` se pueden crear a partir de *matrices de bytes Java*.

```
/**
* Realiza inserción y cierra el store clave-
valor.
*/
void  runExample(){
    final  String  keyString = "/Hola";
    final  String  valueString = "NOSQL World!";
    store.put(Key.fromString(keyString),
    Value.createValue(valueString.getBytes()));
}
```

- El siguiente paso es obtener el valor de la clave que hemos almacenado para comprobar que hemos insertado con éxito el par clave-valor en el store.
- Se usa el método `get` en el identificador del store. Hay muchas variaciones de los métodos `put` y `get` para insertar y recuperar los pares clave-valor en el store.

```
final ValueVersion valueVersion =
store.get(Key.fromString(keyString));
System.out.println(keyString + " " + new
String(valueVersion.getValue().getValue()));
store.close();}
}
```

# Los conceptos básicos de leer y escribir un solo par clave-valor

- El fragmento de código que sigue explora las diferentes formas de crear una clave y almacenar un valor mediante las API de Oracle NoSQL Database.
- En este ejemplo, se crea una clave que se utilizará para hacer referencia a los datos de un bloc de notas de un usuario específico; elegimos un usuario con ID 34271.
  - La parte menor de la clave (la cadena después del guión) es opcional.
  - La inicialización del objeto KVStore se crea llamando al método *getStore* en la clase *KVStoreFactory*.
  - *KVStoreConfig* se crea utilizando el nombre del store al que deseamos conectarnos, así como uno de los equipos host en el clúster de Oracle NoSQL Database y el puerto para contactar en ese equipo host.

```
String notePadKey = "/users/34271/folders/-/notepad";

String valueString = "Un elemento de prueba que no significa nada";

KVStore store = KVStoreFactory.getStore(new KVStoreConfig("kvstore", aStoreHost + ":" +
port)))

Key myKey = Key.fromString(notePadKey);

System.out.println(myKey.getFullPath());

System.out.println(myKey.toString());

Value myValue = Value.createValue(valueString.getBytes());

store.put(myKey,myValue);
```

- En el código siguiente, suponemos que el id de usuario es devuelto por una función de la aplicación que contiene una clase `AuthorizationContext` para recuperar el usuario que ha iniciado sesión actualmente:

```
ArrayList<String> majorList = new ArrayList<String>();
ArrayList<String> minorList = new ArrayList<String>();
int userId = AuthorizationContext.getCurrentUserId();
majorList.add("users");
majorList.add(userId);
majorList.add("folders");
minorList.add("notepad");
Key myKey = Key.createKey(majorList, minorList);
store.put(myKey,myValue);
```

- Ahora echemos un vistazo a cómo usaría la API de base de datos Oracle NoSQL para leer el contenido de la carpeta del bloc de notas de un usuario:

```
ValueVersion valueVersion = store.get(myKey);
String notePadContents = new String(valueVersion.getValue().
getValue());
```

# Durabilidad

- Oracle NoSQL Database codifica la noción de durabilidad en una política que puede ser establecida por el programador para cada llamada a la API que escribe datos en el store.
- En Oracle NoSQL Database, hay dos conjuntos distintos pero relacionados de políticas de durabilidad:
  - *Políticas basadas en reconocimiento de réplicas* Las políticas basadas en reconocimiento definen qué tan estricto debe comportarse el maestro con respecto a cuántas réplicas responden correctamente antes de que el maestro considere la escritura confirmada y responda al *caller* de la API. Hay tres sabores de la durabilidad a nivel de reconocimiento:
    - ***ALL*** Esta es la política basada en el reconocimiento más estricta y duradera y dicta que todas las réplicas deben reconocer las escrituras exitosas antes de que el maestro considere la transacción comprometida. Desde la perspectiva del programador, se puede pensar en esto como una replicación sincrónica, ya que el llamador de la API esperará hasta que todas las réplicas hayan escrito los datos antes de que regrese la llamada a la API.
    - ***SIMPLE_MAJORITY*** Es más estricto y a veces se denomina escritura de quórum. En esta directiva de durabilidad, el maestro replicará de forma asincrónica los datos en todas las réplicas y, a continuación, esperará una respuesta correcta solo de una mayoría, o quórum, de réplicas antes de considerar la transacción confirmada. Por ejemplo, si hay un total de tres nodos (un maestro y dos réplicas), el maestro solo necesita esperar a que una sola réplica responda antes de confirmar la transacción, ya que se han producido un total de dos escrituras, lo que lo convierte en una mayoría.
    - ***NONE*** Esta es la política menos estricta y, desde la perspectiva del programador, esto puede considerarse como una replicación puramente asíncrona. El maestro escribirá los datos localmente, enviará los datos a las otras réplicas e inmediatamente considerará la transacción confirmada sin esperar ninguna respuesta de las réplicas.

- *Políticas basadas en sincronización* Define la garantía básica de que una operación de escritura se ha guardado en almacenamiento persistente. Los niveles altos de sincronización ofrecen una mayor garantía de que la transacción es persistente en el disco, pero a costa de un menor rendimiento. Hay tres tipos de directivas de sincronización y se pueden especificar para el nodo maestro, así como para las escrituras de nodos no maestros:
  - **SYNC** Este es el nivel más estricto de durabilidad e implica la mayor sobrecarga desde una perspectiva de rendimiento. El uso de esta directiva obligará a cada nodo a vaciar la escritura en almacenamiento persistente antes de devolver éxito. Si bien esta política le da al programador un nivel muy alto de confianza en que una escritura nunca se perderá, tiene un costo de rendimiento.
  - **WRITE_NO_SYNC** Este es el siguiente nivel más estricto de durabilidad y hará que cada nodo realice una llamada al sistema que escribirá los datos en la caché del búfer del sistema de archivos, pero no vaciará los datos directamente al almacenamiento persistente. Los datos serán vaciados al almacenamiento persistente por el sistema de archivos de manera asincrónica.
  - **NO_SYNC** Este es el nivel de durabilidad menos estricto y hará que cada nodo escriba los datos en su memoria caché. Los datos se vaciarán en almacenamiento persistente en un punto de control o cuando los datos se desalojan de la memoria caché del nodo.

# Consistencia

- Dado que un almacén clave-valor suele estar compuesto por un clúster de nodos que trabajan juntos de manera distribuida, es posible que el nodo maestro escriba un registro en un fragmento y, posteriormente, lo lea desde otro nodo del fragmento. Dado que hay un desfasado de tiempo entre el momento en que se escribe un registro en el maestro y el tiempo que tarda el registro en transferirse a través de la red a los otros nodos de la partición, es posible que el registro no sea coherente con el maestro si se lee desde un nodo que aún no ha recibido la actualización más reciente del maestro.
- Hay cuatro políticas de consistencia distintas en Oracle NoSQL Database:
  - ***ABSOLUTE*** Esta es la política de consistencia más estricta y dicta que la lectura debe ejecutarse en el nodo maestro del fragmento, lo que garantiza que la versión confirmada más reciente del registro se devuelva al llamador de la API.
    - Si bien el uso de esta política le da al programador una buena garantía para el estado de la lectura, tiene un costo potencial en el rendimiento del sistema, así como en la latencia de lectura.
    - El uso de esta política evitará que el controlador de base de datos NoSQL distribuya la carga de lectura entre todos los nodos del fragmento, lo que posiblemente sobrecargará el nodo maestro y reducirá el rendimiento general del sistema, así como aumentará la latencia de las lecturas en el sistema.
  - ***Time*** Esta es la siguiente política de consistencia más estricta y, cuando es suministrada por el programador junto con un límite de tiempo X y una unidad de tiempo Y, especifica que la lectura se puede realizar contra cualquier nodo en el fragmento siempre que la versión del registro de ese nodo no sea más de X unidades de tiempo de retraso de la versión del registro que se encuentra en el maestro.
    - El controlador de Oracle NoSQL Database siempre es consciente de la topología y puede calcular fácilmente una heurística para saber qué tan lejos está cualquier nodo de su maestro en el fragmento.
    - Como ejemplo de cómo se usaría esta directiva, considere su aplicación de correo electrónico. Digamos que estás leyendo el calendario para el usuario actual. Puede proporcionar una lectura basada en el tiempo, un límite de 500 y una unidad de milisegundos, especificando así que estaría dispuesto a utilizar cualquier nodo en el fragmento para esta lectura, siempre y cuando el registro que está leyendo no esté más de 500 milisegundos rezagados del maestro.

- ***Version*** Esta política de coherencia es al menos tan estricta como la política de coherencia en el tiempo descrita anteriormente, pero se puede utilizar de una manera ligeramente diferente.
  - La coherencia basada en versiones permite al programador proporcionar un objeto *version* a la llamada de lectura y dicta al controlador de Oracle NoSQL Database que puede leer desde cualquier nodo que contenga al menos esta versión de datos y superior.
  - Las versiones en Oracle NoSQL Database son simplemente nociones externalizadas del número de secuencia de registro del sistema de almacenamiento subyacente, y cada inserción en el almacén se etiquetará con un número de secuencia de registro, externalizado a través de las API como una instancia de una clase *Version*.
  - Esta directiva suele ser útil para aquellas aplicaciones que mantienen cierta información de estado sobre objetos insertados o actualizados previamente y pueden utilizar la información de versión guardada para estos objetos como una sugerencia de optimización para el controlador de Oracle NoSQL Database.
- ***NONE_REQUIRED*** Esta política de coherencia le dice al conductor que puede leer el registro de cualquier nodo que crea que es el nodo más óptimo para leer, ya sea que ese nodo tenga o no datos que sean consistentes con su nodo maestro. Por lo tanto, esta política no impone restricciones a la lectura y es la política más óptima que se puede utilizar al leer registros de Oracle NoSQL.
  - Esta política es bastante útil para aquellas cargas de trabajo que tienen latencia muy estricta y favorecen en gran medida la capacidad de devolver algo, aunque pueda estar desactualizado en, digamos, 10 o 15 milisegundos.
  - Vemos este tipo de requisito en el mundo de la publicidad gráfica en línea, donde los editores de contenido en línea han colocado restricciones de latencia extremadamente estrictas en sus proveedores de servicios publicitarios. Estos proveedores están leyendo datos de comportamiento del usuario en un intento de aumentar la probabilidad de que el usuario realmente haga clic en un anuncio que se coloca en el sitio web del editor.

- El siguiente fragmento de código muestra cómo establecer un nivel predeterminado de durabilidad y consistencia, así como cómo se pueden reemplazar estos valores predeterminados en la llamada a la API individual.
  - El primer parámetro define la directiva de sincronización en el nivel de nodo maestro, el segundo parámetro define la directiva de sincronización en el nivel de nodo de replicación y el tercer parámetro configura la directiva de confirmación de replicación.

```
Durability defaultDurability = new Durability(
Durability.SyncPolicy.SYNC, // Master sync
Durability.SyncPolicy.NO_SYNC, // Replica sync
Durability.ReplicaAckPolicy.SIMPLE_MAJORITY);
// Create an instance of the KVStoreConfig class by specifying
the name
// of our store and any machine:port in our cluster of nodes
KVStoreConfig conf = new KVStoreConfig("kvstore",
"a_machine:5000");
conf.setDurability(defaultDurability);
conf.setConsistency(Consistency.NONE_REQUIRED);
store = KVStoreFactory.getStore(conf);
```

- El fragmento de código que sigue creará un par clave-valor y lo insertará en el store en función de una nueva política de durabilidad, que anulará la predeterminada.

```
majorList.add("users");
majorList.add(userId);
majorList.add("folders");
minorList.add("notepad");
Key myKey = Key.createKey(majorList, minorList);
String content = "A test nodepad value";
Value myValue = Value.createValue(st.getBytes());
// Create durability policy to override the durability policy at the
// configuration object level
Durability durability = new Durability(Durability.SyncPolicy.NO_SYNC,
Durability.SyncPolicy.NO_SYNC, Durability.ReplicaAckPolicy.NONE);
try {
store.put(myKey, myValue, null, durability, 0, null);
} catch (DurabilityException de) {
de.printStackTrace();
} catch (RequestTimeoutException re) {
re.printStackTrace();
}
// Override the default consistency policy and specify absolute
// consistency for reading back the record we just wrote
//
ValueVersion vv = store.get(myKey, Consistency.ABSOLUTE, 0, null);
```

# Elegir un modelo

- Tablas
  - Alto nivel de abstracción, simple para modelar, familiar para desarrolladores
  - Índices secundarios manejados por el sistema. Soporte de evolución de tablas.
- JSON
  - Nivel medio de abstracción, es necesario modelar las llaves (como strings), usado para aplicaciones centradas en JSON
  - Vistas de índices manejadas por la aplicación, soporte de evolución de esquemas.
- Clave-valor nativo
  - Bajo nivel de abstracción, es necesario modelar las claves (como strings), la aplicación debe serializar los datos, máxima flexibilidad.
  - Vistas de índices manejadas por la aplicación, evolución de registros (proporcionado por el programador)

# Ventajas de Oracle NoSQL DB

- Registros de tablas distribuidos o “documentos JSON” (el desarrollador elige)
- Registros agrupados localmente (por llave de partición)
- Transacciones ACID
- Mapeo automático de la estructura clave mayor/menor
- Compatible con JSON 2.0
- Soporta evolución de tablas
- CLI para administración de evolución y creación de esquemas