[ENGLISH]

Hello Cloud Gurus,

and welcome to this lecture,

which is a DynamoDB summary, including all of my exam tips.

So remember, DynamoDB is a low latency NoSQL database.

It supports both document and key-value data models.

And the supported document formats are things

like JSON, HTML, and XML.

And when it comes to consistency,

DynamoDB read operations are

eventually consistent by default,

but you can also use

strongly consistent read operations as well.

And DynamoDB transactions allows you

to perform ACID transaction,

so atomic, consistent, isolated, and durable.

In other words, all-or-nothing transactions

and no partial transactions.

And if you think of my example,

if you're making a credit card payment,

you don't want to be in a situation

where your credit card has been charged,

but the person on the other end hasn't received the payment.

You don't want that kind of transaction

to be only partially completed.

It has to be all or nothing.

Data is stored in tables as items and attributes.

And when we think of items,

just think of a row in the table.

And when we think of attributes,

I want you to think of columns in the table.

There are two types of primary key supported,

and, of course, the primary key must be a unique identifier

for your data items in your DynamoDB table.

So there are two types of primary key.

Firstly, we have the partition key,

and an example of a good partition key is something like

a customer ID, product ID,

or a vehicle registration number, et cetera.

And as long as your partition key is a unique identifier

for the items in your table,

then you are good to go.

But if that is not the case,

then you will need to use a composite key,

which is a combination of a partition key and a sort key.

For example, think of a table storing data

about users posting messages in a forum,

and the same user ID can appear multiple times in the data

because you'd expect a user to post multiple times

in the same forum.

So we need to use a composite key,

something like a user ID plus a sort key of a timestamp

to give you a uniquely identifying primary key for our data.

Moving on to Access Control for DynamoDB.

You can configure Fine-Grained Access Control

with Identity Access Management,

and there's this IAM condition parameter

called dynamoDB:LeadingKeys,

which allows users to access only the items

where the partition key value matches their user ID.

So this means that the users can only access their own data

instead of all of the data in the table.

Onto secondary indexes.

And indexes enable fast queries on specific data columns

within your DynamoDB table,

and they provide a different view of your data

based on an alternative partition key

or an alternative partition and sort key combination.

And it's important to understand the differences

between the two types of secondary index.

So firstly, we have the local secondary index,

and a local secondary index uses the same partition key

but a different sort key to your original table.

And it must be created when you first create your table.

You cannot create it later on.

And then we have the global secondary index,

and this is a little bit more flexible.

So this allows you to define a different partition key

and a different sort key to your original table,

and it can be created at any time,

so you don't need to create it when you create the table.

Onto DynamoDB queries,

and a query allows you to find items in a table

using only the primary key attribute.

And you provide the primary key name

and a distinct value to search for.

The results of a query are always sorted

in ascending order by the sort key if there is one.

And if you'd like to reverse the order,

you can use the ScanIndexForward parameter,

and you just set the ScanIndexForward parameter to false.

And this reverses the order of query results.

And strangely, even though they've called it

ScanIndexForward, it only applies to queries,

so you cannot use this with scans, it's just for queries.

And speaking of scans, a scan operation examines

every item in the table,

and by default, it returns all data attributes,

and you can use the ProjectionExpression parameter

to refine your results,

so if you only wanted to see the customer ID

and the email address,

then you would use the ProjectionExpression parameter

to show just the customer ID and the email address

instead of all the other data attributes as well.

And if you would like to make your scans more efficient,

there's a few things that you can do.

So firstly, to reduce the impact of a query or a scan,

you can set a smaller page size,

which uses fewer read operations.

You can isolate the scan operations to specific tables

and segregate them from your mission-critical traffic.

You can also try parallel scans

rather than the default sequential scan,

but generally, a query is more efficient than a scan.

So if you possibly can, or if you have a choice,

then use a query if you can rather than a scan.

Because a scan, it always dumps the entire data set.

And then after that,

you can either use the ProjectionExpression,

or you can apply a filter,

but before you do that, it always dumps the entire table.

So it's never going to be as efficient

as querying based on a specific attribute.

So if you possibly can, avoid a scan

and design your tables in such a way

that you can use Query, Get or BatchGetItem APIs

rather than using the scan API.

Onto provisioned throughput, and just remember,

provisioned throughput is measured in capacity units,

and we've got reed capacity units and write capacity units.

So with write capacity units,

each write capacity unit gives you

one one-kilobyte write per second.

And then with read capacity units,

you get one four-kilobyte

strongly consistent read per second.

Whereas for eventually consistent reads,

you get two four-kilobyte

eventually consistent reads per second.

So with eventually consistent, that is actually the default,

and you get double the throughput

then you would get for a strongly consistent read.

And there is also an On-Demand pricing model as well.

And it's important to understand the difference

and know when you should use each pricing model.

So with On-Demand Capacity, this is great

if you have unpredictable application traffic,

and you would like to use a pay-per-use model.

Whereas you should use provision capacity when your read

and write capacity requirements can be forecasted,

and application traffic is either consistent,

or it's only increasing gradually.

And Provisioned Capacity is a good one to use

if you want to be more in control of your costs

rather than having your DynamoDB table

scale automatically on-demand,

meaning that your costs will be increasing as well.

Whereas with Provisioned Capacity,

you set the capacity that you're going to be using,

and you will have no surprises on your AWS bill.

Moving on to DynamoDB Accelerator,

and it's also known as DAX.

And DAX is an in-memory cache providing in-memory caching

for your DynamoDB tables,

and this improves the response time

for eventually consistent reads only,

so it doesn't work with strongly consistent reads,

and it doesn't work with writes either.

And when a write operation occurs to your DynamoDB table,

the data is written to the cache

and the backend store at the same time,

And then you point your application

or your API calls at the DAX cluster instead of your table.

So this takes the workload away from your DynamoDB table,

and if the item that you are querying for is in the cache,

then DAX is going to return it.

However, as I said,

it's not suitable for write-intensive applications

or applications which require strongly consistent reads

because it's not going to improve

either of those situations.

Onto DynamoDB TTL or Time to Live,

and TTL defines an expiry time for your data.

And once the data has expired,

then it's going to be marked for automatic deletion.

And this is great for removing irrelevant

or old or expired data,

for example, session data, event logs,

and temporary data that your application no longer needs.

And this is gonna help you save money

by reducing the cost of your table

because it's automatically removing data

which is no longer relevant and no longer used

by your application.

Onto DynamoDB Streams,

and a DynamoDB Stream is a sequence of modifications

to your DynamoDB table.

So it's a time-ordered sequence

of item level modifications in your table.

And it's similar to an audit trail,

recording all the changes

that are happening in your DynamoDB table.

The data is encrypted and stored for 24 hours only,

and it's great as a Lambda event source.

So you can create applications that take actions based

on events in your DynamoDB table.

And if your table is suddenly receiving

a large amount of requests,

then you might see a ProvisionedThroughputExceeded error,

and this means that the number of requests

to your DynamoDB table is too high

for the read and write capacity units

that you have configured.

So there are too many read and writes coming through,

and your table is becoming overloaded.

So what can you do?

Well, this is where exponential backoff comes in,

and exponential backoff improves flow control

by retrying requests using progressively longer waits.

So it's going to back off the requests

and retry a few seconds later to see

if the request is successful.

And if you've written your own application code.

then it's a good idea to configure your application

to use exponential backoff to avoid this kind of error.

However, if you are using the AWS SDK

then exponential backoff is a feature of every AWS SDK,

applying to many services within AWS,

so S3, CloudFormation, SES, et cetera.

And it's only really

if you've been writing your own application code

that you will need to be aware

to explicitly use exponential backoff

to avoid this kind of error.

Remember the AWS CLI commands

that you can use to make API calls to the DynamoDB API,

and you should understand when to use each one.

And be aware

that the correct Identity and Access Management permissions

are required to make an API call.

And they're all pretty well named,

so they are mostly self-explanatory,

so there's create-table, put-item,

get-item, update-item,

update-table, and list-tables.

So these are the CLI commands,

and they correspond to these API calls.

So in order to run the command,

make sure your Identity and Access Management user has

permissions for the underlying API call.

And then there's describe-table, scan, and query,

delete-item and delete-table.

And here's the corresponding API calls

And that is a summary of our DynamoDB section,

and that's the end of this lecture.

So well done, everybody.

You've got to the end of the DynamoDB section.

If you have any questions, please do let me know.

Otherwise, I'll see you in the next section.

Thank you.

[SPANISH]

Hola gurús de la nube,

y bienvenidos a esta conferencia,

que es un resumen de DynamoDB, que incluye todos mis consejos para el examen.

Así que recuerde, DynamoDB es una base de datos NoSQL de baja latencia.

Admite modelos de datos de documentos y valores clave.

Y los formatos de documentos admitidos son cosas

como JSON, HTML y XML.

Y cuando se trata de consistencia,

Las operaciones de lectura de DynamoDB son

eventualmente consistente por defecto,

pero también puedes usar

operaciones de lectura fuertemente consistentes también.

Y las transacciones de DynamoDB le permiten

para realizar la transacción ACID,

tan atómico, consistente, aislado y duradero.

En otras palabras, las transacciones de todo o nada

y sin transacciones parciales.

Y si piensas en mi ejemplo,

si está haciendo un pago con tarjeta de crédito,

no quieres estar en una situación

donde se ha cargado su tarjeta de crédito,

pero la persona del otro lado no ha recibido el pago.

No quieres ese tipo de transacción.

para ser solo parcialmente completado.

Tiene que ser todo o nada.

Los datos se almacenan en tablas como elementos y atributos.

Y cuando pensamos en artículos,

solo piense en una fila en la tabla.

Y cuando pensamos en atributos,

Quiero que pienses en las columnas de la tabla.

Se admiten dos tipos de clave principal,

y, por supuesto, la clave principal debe ser un identificador único

para sus elementos de datos en su tabla de DynamoDB.

Así que hay dos tipos de clave principal.

En primer lugar, tenemos la clave de partición,

y un ejemplo de una buena clave de partición es algo como

una identificación de cliente, identificación de producto,

o el número de matrícula del vehículo, etcétera.

Y siempre que su clave de partición sea un identificador único

por los artículos de tu mesa,

Entonces, ya puedes irte.

Pero si ese no es el caso,

entonces necesitará usar una clave compuesta,

que es una combinación de una clave de partición y una clave de clasificación.

Por ejemplo, piense en una tabla que almacena datos

sobre los usuarios que publican mensajes en un foro,

y el mismo ID de usuario puede aparecer varias veces en los datos

porque esperarías que un usuario publicara varias veces

en el mismo foro.

Así que necesitamos usar una clave compuesta,

algo así como una identificación de usuario más una clave de clasificación de una marca de tiempo

para darle una clave primaria de identificación única para nuestros datos.

Pasando al control de acceso para DynamoDB.

Puede configurar el control de acceso detallado

con gestión de acceso a la identidad,

y está este parámetro de condición de IAM

llamado dynamoDB:LeadingKeys,

que permite a los usuarios acceder solo a los elementos

donde el valor de la clave de partición coincide con su ID de usuario.

Esto significa que los usuarios solo pueden acceder a sus propios datos.

en lugar de todos los datos de la tabla.

En índices secundarios.

Y los índices permiten consultas rápidas en columnas de datos específicas

dentro de su tabla de DynamoDB,

y proporcionan una vista diferente de sus datos

basado en una clave de partición alternativa

o una combinación alternativa de partición y clave de clasificación.

Y es importante entender las diferencias.

entre los dos tipos de índice secundario.

En primer lugar, tenemos el índice secundario local,

y un índice secundario local usa la misma clave de partición

pero una clave de clasificación diferente a su tabla original.

Y debe crearse cuando crea su tabla por primera vez.

No puedes crearlo más tarde.

Y luego tenemos el índice secundario global,

y esto es un poco más flexible.

Entonces esto le permite definir una clave de partición diferente

y una clave de clasificación diferente a su tabla original,

y se puede crear en cualquier momento,

por lo que no necesita crearlo cuando crea la tabla.

En consultas de DynamoDB,

y una consulta le permite encontrar elementos en una tabla

usando solo el atributo de clave principal.

Y usted proporciona el nombre de la clave principal

y un valor distinto para buscar.

Los resultados de una consulta siempre se ordenan

en orden ascendente por la clave de ordenación, si la hay.

Y si desea invertir el orden,

puede usar el parámetro ScanIndexForward,

y acaba de establecer el parámetro ScanIndexForward en falso.

Y esto invierte el orden de los resultados de la consulta.

Y extrañamente, aunque lo hayan llamado

ScanIndexForward, solo aplica para consultas,

por lo que no puede usar esto con escaneos, es solo para consultas.

Y hablando de escaneos, una operación de escaneo examina

cada elemento de la mesa,

y por defecto, devuelve todos los atributos de datos,

y puedes usar el parámetro ProjectionExpression

para refinar sus resultados,

así que si solo quisiera ver la identificación del cliente

y la dirección de correo electrónico,

entonces usaría el parámetro ProjectionExpression

para mostrar solo el ID del cliente y la dirección de correo electrónico

en lugar de todos los demás atributos de datos también.

Y si desea que sus escaneos sean más eficientes,

hay algunas cosas que puedes hacer.

En primer lugar, para reducir el impacto de una consulta o un escaneo,

puede establecer un tamaño de página más pequeño,

que utiliza menos operaciones de lectura.

Puede aislar las operaciones de escaneo en tablas específicas

y sepárelos de su tráfico de misión crítica.

También puede probar escaneos paralelos

en lugar del escaneo secuencial predeterminado,

pero generalmente, una consulta es más eficiente que un escaneo.

Entonces, si es posible, o si tiene otra opción,

luego use una consulta si puede en lugar de un escaneo.

Debido a que un escaneo, siempre vuelca todo el conjunto de datos.

Y luego después de eso,

puede usar ProjectionExpression,

o puedes aplicar un filtro,

pero antes de hacer eso, siempre vuelca toda la tabla.

Así que nunca va a ser tan eficiente

como consultas basadas en un atributo específico.

Entonces, si puede, evite un escaneo

y diseña tus mesas de tal manera

que puede usar las API Query, Get o BatchGetItem

en lugar de usar la API de escaneo.

Sobre el rendimiento aprovisionado, y recuerde,

el rendimiento aprovisionado se mide en unidades de capacidad,

y tenemos unidades de capacidad de lengüeta y unidades de capacidad de escritura.

Entonces, con las unidades de capacidad de escritura,

cada unidad de capacidad de escritura te da

una escritura de un kilobyte por segundo.

Y luego con unidades de capacidad de lectura,

obtienes uno de cuatro kilobytes

Lectura fuertemente consistente por segundo.

Mientras que para lecturas eventualmente consistentes,

obtienes dos de cuatro kilobytes

eventualmente lecturas consistentes por segundo.

Entonces, con eventualmente consistente, ese es en realidad el valor predeterminado,

y obtienes el doble de rendimiento

entonces obtendría una lectura fuertemente consistente.

Y también hay un modelo de precios bajo demanda.

Y es importante entender la diferencia

y sepa cuándo debe usar cada modelo de precios.

Entonces, con capacidad bajo demanda, esto es genial

si tiene un tráfico de aplicaciones impredecible,

y le gustaría utilizar un modelo de pago por uso.

Mientras que debe usar la capacidad de provisión cuando lee

y se pueden pronosticar los requisitos de capacidad de escritura,

y el tráfico de la aplicación es consistente,

o sólo está aumentando gradualmente.

Y la capacidad aprovisionada es buena para usar

si desea tener un mayor control de sus costos

en lugar de tener su tabla de DynamoDB

escalar automáticamente bajo demanda,

lo que significa que sus costos también aumentarán.

Mientras que con la Capacidad Aprovisionada,

usted establece la capacidad que va a utilizar,

y no tendrás sorpresas en tu factura de AWS.

Pasando a DynamoDB Accelerator,

y también se conoce como DAX.

Y DAX es un caché en memoria que proporciona almacenamiento en caché en memoria

para sus tablas de DynamoDB,

y esto mejora el tiempo de respuesta

solo para lecturas eventualmente consistentes,

por lo que no funciona con lecturas fuertemente consistentes,

y tampoco funciona con escrituras.

Y cuando se produce una operación de escritura en su tabla de DynamoDB,

los datos se escriben en el caché

y la tienda backend al mismo tiempo,

Y luego apuntas tu aplicación

o sus llamadas API en el clúster DAX en lugar de su tabla.

Así que esto le quita la carga de trabajo a su tabla de DynamoDB,

y si el elemento que está consultando está en el caché,

entonces DAX lo devolverá.

Sin embargo, como dije,

no es adecuado para aplicaciones de escritura intensiva

o aplicaciones que requieren lecturas fuertemente consistentes

porque no va a mejorar

cualquiera de esas situaciones.

En DynamoDB TTL o Time to Live,

y TTL define un tiempo de caducidad para sus datos.

Y una vez caducados los datos,

luego se marcará para su eliminación automática.

Y esto es genial para eliminar elementos irrelevantes

o datos antiguos o caducados,

por ejemplo, datos de sesión, registros de eventos,

y datos temporales que su aplicación ya no necesita.

Y esto te ayudará a ahorrar dinero.

reduciendo el costo de su mesa

porque está eliminando datos automáticamente

que ya no es relevante y ya no se usa

por su aplicación.

en flujos de DynamoDB,

y un DynamoDB Stream es una secuencia de modificaciones

a su tabla de DynamoDB.

Entonces es una secuencia ordenada en el tiempo

de modificaciones de nivel de artículo en su tabla.

Y es similar a un registro de auditoría,

grabando todos los cambios

que están sucediendo en su tabla de DynamoDB.

Los datos se cifran y almacenan solo durante 24 horas,

y es excelente como fuente de eventos de Lambda.

Para que pueda crear aplicaciones que realicen acciones basadas

en los eventos de su tabla de DynamoDB.

Y si tu mesa de repente recibe

una gran cantidad de solicitudes,

entonces es posible que vea un error ProvisionedThroughputExceeded,

y esto significa que el número de solicitudes

a su tabla de DynamoDB es demasiado alto

para las unidades de capacidad de lectura y escritura

que has configurado.

Así que hay demasiadas lecturas y escrituras llegando,

y tu mesa se está sobrecargando.

¿Entonces que puedes hacer?

Bueno, aquí es donde entra en juego el retroceso exponencial,

y el retroceso exponencial mejora el control de flujo

al volver a intentar las solicitudes usando esperas progresivamente más largas.

Así que va a retroceder en las solicitudes.

y vuelva a intentarlo unos segundos más tarde para ver

si la solicitud tiene éxito.

Y si ha escrito su propio código de aplicación.

entonces es una buena idea configurar su aplicación

utilizar el retroceso exponencial para evitar este tipo de error.

Sin embargo, si está utilizando AWS SDK

entonces el retroceso exponencial es una característica de cada SDK de AWS,

aplicando a muchos servicios dentro de AWS,

entonces S3, CloudFormation, SES, etcétera.

Y es solo realmente

si ha estado escribiendo su propio código de aplicación

que tendrás que ser consciente

para usar explícitamente el retroceso exponencial

para evitar este tipo de error.

Recuerde los comandos de la CLI de AWS

que puede usar para realizar llamadas de API a la API de DynamoDB,

y usted debe entender cuándo usar cada uno.

y ser consciente

que los permisos correctos de gestión de acceso e identidad

son necesarios para realizar una llamada a la API.

Y todos están bastante bien nombrados,

por lo que en su mayoría se explican por sí mismos,

así que hay una tabla de creación, un artículo de venta,

obtener elemento, actualizar elemento,

tabla de actualización y tablas de lista.

Así que estos son los comandos CLI,

y corresponden a estas llamadas API.

Así que para ejecutar el comando,

asegúrese de que su usuario de Gestión de acceso e identidad tenga

permisos para la llamada a la API subyacente.

Y luego está la tabla de descripción, el escaneo y la consulta,

eliminar elemento y eliminar tabla.

Y aquí están las llamadas API correspondientes

Y ese es un resumen de nuestra sección de DynamoDB,

y ese es el final de esta conferencia.

Muy bien hecho, todos.

Has llegado al final de la sección de DynamoDB.

Si tiene alguna pregunta, por favor hágamelo saber.

De lo contrario, te veré en la siguiente sección.

Gracias.