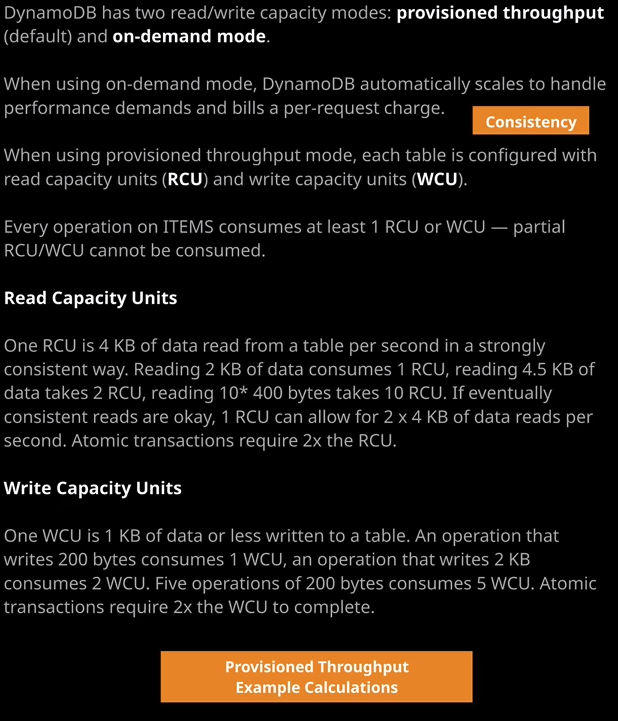
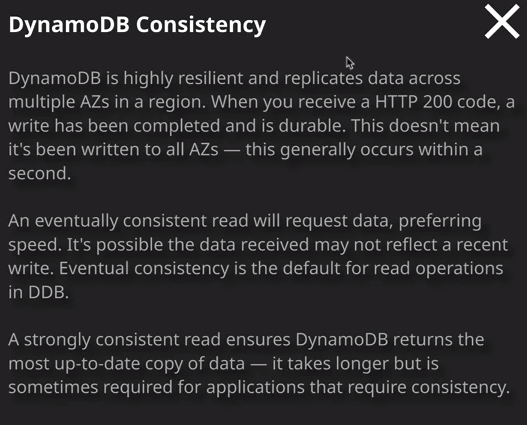
**performance and billing in DynamoDB.**



Now before we start, I want to talk about **partitions**. In the last lesson, I talked about how **DynamoDB was a serverless database service.** You didn't get exposure to any servers or any infrastructure but as we all know, **serverless does not mean a lack of servers. Even a serverless products such as DynamoDB, does need some infrastructure in order to deliver the service and the way that DynamoDB handles this service delivery is via an entity called the partition**. **Partitions you can think of as low-level entities which store your data. So DynamoDB achieves the performance that it does, the high levels of performance, by splitting your data across partitions. When you write items to DynamoDB so when I created this weather data table and wrote all of these items to DynamoDB, they're being stored in a particular partition. The way that DynamoDB handles this is for a specific value of the partition key that's passed to something known as a hashing function and the hashing function spits out the ID of a partition to use. So you always know for a given value for a partition key, which partition that data will be stored in.** So DynamoDB knows that when I'm writing data for station ID 0001 it's always going to go on the same partition and when I want to retrieve data using 0001 as the value for the partition key, it knows exactly where to look.

Now, **depending on a few factors, generally, when you create a table, it starts off with a single partition but that single partition can grow to many more, depending on how much data you store in that table and how much performance, known as capacity, that you require. DynamoDB will increase the number of partitions that a table uses as more demands placed on that table.** Now we'll talk about the different ways that you can influence table performance later in this lesson but **any configuration that you make on a table, any demands for a certain level of read and write capacity is actually distributed across all of the partitions that make up that table. So if you allocate a certain level of performance for a table with four partitions, then each partition only has 1/4 or 25% of the performance of the table.** Now why this matters let's assume that the weather data table I created in the previous lesson had two partitions. If it had two partitions, it could be that weather stations 0001 and three are in partition one and partition two holds the second weather station 0002. Now, whatever performance the table has been given for reads and writes then each partition because they're two partitions can only ever get 50% of the total. Now, as I mentioned earlier, because each value for the partition key or in this case every weather station will always be stored on the same partition. It means that if all I'm doing is reading or writing one single partition key value so one weather station I can only ever get the maximum performance that's allocated to the partition, not to the table. Now, knowing the number of partitions that are allocated and being able to effectively manage performance is something that you need more towards the professional certification levels in the associate but I always like you to be aware of exactly how things work from an infrastructure perspective. **So when you're allocating performance for DynamoDB tables, make sure you keep in mind that what you're actually doing is allocating it to partitions, not to tables.**

Now, I mentioned earlier in this lesson that **DynamoDB replicates your data across at least three availability zones. Each availability zone has a copy of the data for each of those partitions, so a partition is replicated into three different availability zones, and these copies are stored on things called nodes**. **For each partition, there's a single leader node and two additional non leader nodes**. Now, **when you write data to DynamoDB, it gets written to the leader node. Data is written, and then it's replicated from the leader node to the other non leader nodes**.



**When you read data you can do it in one of two different ways, and this is known as data consistency. When you request data, you can either read in what's known as a strongly consistent way, which means that when you make the request to get data, then it gets that via the leader node. So essentially the leader node is always going to be correct. That's the node that data gets written to and if you insist on strongly consistent reads then the leader node is also used to retrieve data.** **Now you'd use strongly consistent reads if you have an application that always needs guaranteed access to the most recently updated data.**

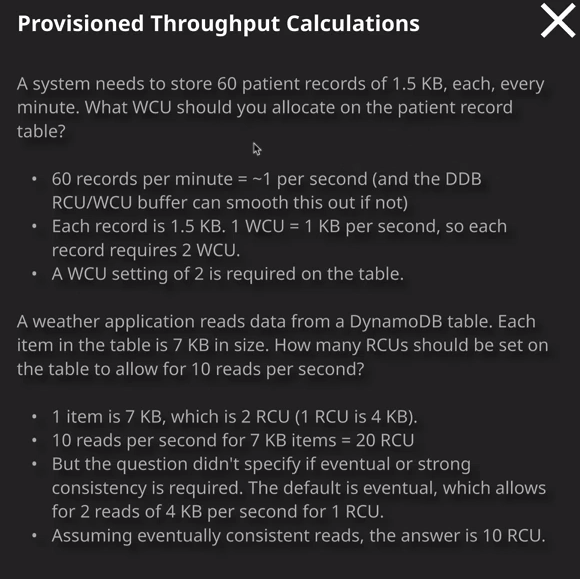
**Another option is what's known as eventually consistent reads and with eventually consistent reads, it essentially means that you can read the data from any of the three nodes.** Now, in some cases, this will still be the leader node meaning the data that you'll receive is always up to date. It could also be a non leader node and if it's a non leader node then the data, could also still be up to date, because by the point that you're making the get request. When you're reading the data from that table, it's replicated to all of the non leader nodes. But it is possible that **with eventually consistent reads, you could read from a non leader node that has not yet got an up to date copy of the data and that's what I mean when I say eventual consistency.** So with eventually consistent reads, there is a tiny chance that if you read an item immediately after writing it, you might get old data. That's what eventually consistent read means.

Now when you read from DynamoDB, you can pick eventually consistent or strongly consistent. **The default is to use eventually consistent reads, and the reason for the default is that they're actually cheaper. Any costs for AWS and any calculations that you'll perform are based on strongly consistent reads and eventually consistent are half the cost. So where possible, use eventually consistent reads,** because you'll be able to get more performance for the same amount of spend from DynamoDB.

Now there are two capacity modes available inside DynamoDB for a given table. So if I click on the capacity tab, then we've got two different selections that we can make for the read and write capacity mode. **We've got on-demand and we've got provisioned**. Now **you can switch between these two different modes once every 24 hours, and the reason for this delay is that making this change does make some underlying infrastructure adjustments and so DynamoDB limits your ability to make that change to once every 24 hours.**

Now, with **on-demand performance, you don't have to worry about the performance on the table. You pay a certain amount per million writes and per million reads in addition to data transfer. DynamoDB in the background will handle everything else for you. It's nice and simple and if you use indexes**, which we'll talk about later in this topic, the same logic applies to those. **Now you'd use on-demand for any new applications or applications where the workload is too complex to forecast. If you're developing an application that's multi tenant and uses pay per use pricing then by using on-demand you make sure that your costs a directly aligned to the income that you're generating from the application**. So you make sure that wherever price that you sell your application to your customers for you've included an appropriate amount of on-demand pricing for your underlying database.

Now **the other mode of read and write capacity is known as provisioned and with provisioned throughout you specify a read and write capacity value on a table. We've got auto scaling**, which I'll talk about in a minute. But if I untick read and write capacity auto scaling then I'm allowed to specify a read capacity unit value and a write capacity unit value for this table. Now I mentioned earlier in this topic that an item in DynamoDB could be up to 400 kilobytes in size. Now it could be 400 kilobytes. It could be 10 bytes, 400 bytes, or anywhere in between. The important thing to understand **is that every time you run an operation which reads or writes data from or to a DynamoDB table, you consume capacity units. A single read operation will consume one capacity unit for every four kilobytes of data if read in an immediately or strongly consistent way or half that 0.5 capacity units for every four kilobytes of eventually consistent read data.** Now that's a minimum. **If you attempt to read any less than four kilobytes in a single operation, it will always consume naught .5 for eventually consistent or one for strongly consistent. Now write operations follow a similar logic. A write operation will consume one write capacity unit for every one kilobyte of data written. Now logically, with writes, there's no different consistency modes a write is a write.**



Now if you read or write items bigger than those values. So, for example, if you write items bigger than one kilobyte, for example, five kilobytes or 10 kilobytes, then it would consume five WCU or 10 WCU. Now, if you're performing lots of operations which operate on small items, so items that are smaller than these minimums, then you're still going to consume one unit for each of those operations. If you run 10 get item operations, that only consume a tiny amount of data, maybe 10 bytes or 20 bytes. Then you're still going to consume 10 read capacity units, so wherever possible, you need to be running operations, **which can operate on larger blocks of items and do so as one unit.** And if you do that, you could be more efficient with your read and write capacity units. So let's look at a couple of examples. So let's say that we have a system that needs to store 60 patient records into DynamoDB every minute, so that's one per second and each of those records is 1.5 kilobytes in size. Now, DynamoDB does include a buffer that can help smooth this out. So for exam questions, you can assume that if you're given a total of 60 records per minute, you can stretch this out and say one per second. Now we know that each of these patient records is 1.5 kilobytes and we know that one WCU is one kilobyte, so each patient record represents two write capacity units. Now, a write capacity unit is actually a per second metric, so one write capacity unit allows you to write one kilobyte per second to that table. So if each record and we're doing one per second uses two write capcity units because it's 1.5 kilobytes that means we need a setting of two on the table. Let's look at another calculation. We've got a weather application, which reads data from a DynamoDB table. Each item on the table is seven kilobytes in size. How many read capacity units should be set on the table to allow for 10 reads per second? So one read or one item is seven kilobytes in size, and a read capacity unit is four kilobytes. So one read capacity unit allows for four kilobytes of read every second, so one item is seven kilobytes. That means we need two read capacity units per item. We know what we need to do 10 of those seven kilobyte reads every second. If each of the items, because it's seven kilobytes, if each of those uses two read capacity units then 10 reads per second equals 20 RCU. Now this is where it gets interesting. The question didn't specify if it needs eventual consistency or strong consistency. The default is eventual consistency, and if you don't see it mentioned in the exam question, if it doesn't specify explicitly whether it needs eventual or strongly consistent reads than you can assume eventual. So it would need 20 RCU for strongly consistent but it would be half of that for eventual consistent, and that's 10 RCU. Now this might seem fairly theory heavy, and it probably is for the solutions architect associate exam. I want you to understand, though, what an RCU is and what the WCU is because if you're ever having to create a DynamoDB table and you need to use provisioned capacity, you need to understand exactly how to plan this. If you're using on-demand capacity, you don't need to worry about this at all because it's a per operation charge per reads and per writes. With provisions though you need to specify appropriate levels for the read capacity and the write capacity. Now, **generally speaking, provisioned capacity is cheaper than on-demand. So the trade off you're essentially making is if you're unsure about your workload or if you're using a per usage charge to your clients, then you can use on-demand knowing that it's going to be slightly more expensive but you've got that aligned with the income that you're getting from your application.** If you focused on costing you wanting to make it as cheap as possible and you understand your workloads then you can use provisioned capacity. But in order to do that, you need to be aware of exactly how to calculate this.

Now, one last thing that I want to mention before we finish up this lesson **DynamoDB does now include auto scaling capability, which means that you don't have to explicitly specify the read capacity units and the write capacity units. You can enable auto scaling. You can define a minimum and maximum for each of the read and write elements of a table, and DynamoDB will automatically adjust the read and write capacity allocated to a table based on those demands**. Now, strictly speaking, this is beyond what you need for the solutions architect associate exam. But I have had students who've had questions based on DynamoDB capacity and so I want to give you all the tools that you'll need going into the exam to be able to confidently answer questions. Now, I've only given two examples of how to do these provisioned throughput calculations. So what I recommend is use The Orion Papers, review each of these examples, and make sure you understand exactly how it works. But from a content perspective, that's everything that I wanted to cover in this lesson. So go ahead, mark the lesson as complete and when you're ready, you can join me in the next lesson where I'm going to be talking about DynamoDB streams and triggers.