5 Strategies I Use To Save Big On Costs With DynamoDB

medium.com/towards-aws/5-strategies-i-use-to-save-big-on-costs-with-dynamodb-94f976e5d090

Uriel Bitton 9 September 2024

7 proven strategies that help me and will help you potentially save thousands of dollars a month (at scale)

Article cover image

Have you stopped using DynamoDB or heard of someone doing so because of its high costs?

This is a story I've heard more than once or twice.

Just last week, I was chatting with a LinkedIn connection who was telling me about his company's decision to migrate some of its database off of DynamoDB.

The sole reason?

High costs.

But he admitted to me that the company (or the database developers) wasn't well acquainted with how to optimize for costs nor how to design tables with costs in mind.

This led me to believe many companies out there as well are building databases unaware of how to optimize them for costs.

As this becomes a bigger problem for more and more businesses, I saw an opportunity to create a newsletter educating developers and larger teams on frugality in the cloud (I'll leave a link to the newsletter below).

This lack of knowledge of optimizing DynamoDB for costs also led me to write this article.

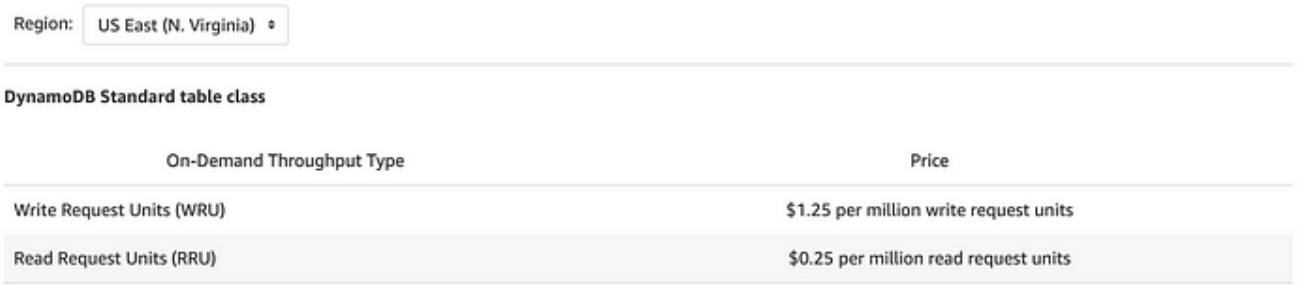
In an effort to help those struggling with costs in DynamoDB, I layout 5 strategies you can use right now to save big on your monthly database costs.

1. Choosing Lower Cost Regions

Not every region has the same table costs.

For example, the costs of read and write request units in the us-east-1 region is significantly less expensive than the sa-east-1 (South America) region.

Here is a price comparison:



Region: US East (N. Virginia)

DynamoDB Standard table class

On-Demand Throughput Type	Price
Write Request Units (WRU)	$1.25 per million write request units
Read Request Units (RRU)	$0.25 per million read request units

costs for dynamodb write/read request units in DynamoDB — us-east-1 region

Region: South America (Sao Paulo)

DynamoDB Standard table class

On-Demand Throughput Type	Price
Write Request Units (WRU)	$1.875 per million write request units
Read Request Units (RRU)	$0.375 per million read request units

costs for dynamodb write/read request units in DynamoDB — sa-east-1 region

That's an increase of more than 60 cents per million write/read request units.

The price difference is also evident in all other data metrics.

Choosing a table that can strike a balance between proximity and price can be beneficial.

2. Setting TTLs on temporary data

Oftentimes you will have to store temporary data in your DynamoDB tables.

These can be any of the following:

- temporary access codes
- log data
- user session data
- expirable tokens
- old notifications
- Any data that is not needed after x amount of years

Storing this data in DynamoDB can quickly become costly, especially at scale.

The problem is, that identifying and deleting this data will ironically end up costing you as well (to query/scan for it).

However, by using DynamoDB's TTL feature — time to live — the data will automatically be deleted by DynamoDB without incurring any costs at all.

TTLs are simple to set up on your table items. I wrote an extensive article about this here.

3. Keep attribute names short

Not long ago I wrote an article about how shortening attribute names could help you save costs in DynamoDB.

While this remains a more drastic measure to squeeze out charges, it can be quite effective at larger scales.

The idea behind this is that DynamoDB charges you for the size of the data returned from your queries.

If your queries return smaller data you are charged less for the same query.

One way of optimizing this is by shortening your attribute names.

For example, "lastDateSubmittedByUser" could become a gluttonous attribute name when used on every item in a table.

A good strategy would be renaming the attribute name to "LDSU" and then renaming it back to the original name on the front end.

4. Use Infrequent Access Tables

DynamoDB offers two table classes to help you optimize your table's costs.

These are:

- Standard table class
- Standard-Infrequent Access table class

The standard table class is the default table that is designed to balance storage and throughput costs.

On the other hand, the standard-infrequent access table class offers up to 60% lower storage costs and 25% higher read/write costs than the standard table.

The takeaway here is to choose the infrequent table class for infrequently accessed data when you can to save on costs.

5. Don’t store large objects

DynamoDB was never designed to store large objects.

Large objects can consist of blob files, long text, or objects with many attributes.

DynamoDB promises single-digit latency queries and for that reason, data queried is always limited to 1MB.

Furthermore, a single item’s maximum size cannot surpass 400 KBs.

Instead, large objects should be stored in Amazon S3 or compressed if they absolutely need to be stored in your DynamoDB table.

Get an email whenever Uriel Bitton publishes.

Get an email whenever Uriel Bitton publishes. By signing up, you will create a Medium account if you don't already have…

medium.com

Summary

You can save significantly on costs by using the following 5 simple strategies.

This includes choosing lower-cost regions, leveraging TTLs, shortening attribute names, using infrequent access tables, and avoiding storing large objects.

📖 P.S. I’m creating a free email course on DynamoDB that will teach you concepts and design patterns not taught in any existing course out there. Signup for free here:

https://www.buildawaitlist.com/waitlist/dynamodb-course

👋 *My name is Uriel Bitton and I'm committed to helping you master DynamoDB, Serverless, and AWS.*

🚀 *If you want to learn how to build serverless, scalable, and resilient applications, you can subscribe to get my posts by email:*

https://medium.com/@atomicsdigital/subscribe

Thanks for reading and see you in the next one!