**What is this course about?**

See why you should take this course and what you will learn.

**WE'LL COVER THE FOLLOWING**

* + [What to choose?](https://www.educative.io/courses/good-parts-of-aws/m71p7xAr2qn#what-to-choose)
  + [What to expect?](https://www.educative.io/courses/good-parts-of-aws/m71p7xAr2qn#what-to-expect)

**What is this course about?**

This is not AWS’s typical reference course. It doesn’t cover all of AWS or all its quirks. Instead, we want to help you realize which AWS features you’d be foolish not to use. Features for which you almost never need to consider alternatives. Features that have passed the test of time by being at the backbone of most things on the Internet.

**What to choose?**[#](https://www.educative.io/courses/good-parts-of-aws/m71p7xAr2qn#what-to-choose)

Making technical choices can be overwhelming. As developers, we have to make many choices from what seems like unlimited options. We have to choose:

* a programming language
* an application framework
* a database
* a cloud vendor
* software dependencies
* and many other things

AWS alone offers about 150 services, all with their own set of options and features, sometimes even overlapping with each other.

AWS Top 15 services

In this course, we will reveal a technique we use ourselves to help make reliable technical choices without getting paralyzed in the face of so many options. This technique will be presented in the context of AWS, but it can be generalized to any other technical decision.

**Q**

Which of the following is the central application in the AWS fleet of services?

**Your Answer**

**A)**

Amazon Elastic Compute Cloud

**B)**

Amazon Simple Queue Service

**C)**

Amazon Simple Notification Service

**D)**

Amazon Simple Storage System

**Great, you got it right!**

Retake Quiz

Found the above quiz hard? Don’t worry, you will get to know a lot about AWS services and best practices after finishing this course.

**What to expect?**[#](https://www.educative.io/courses/good-parts-of-aws/m71p7xAr2qn#what-to-expect)

This is an opinionated course. We only cover topics we have significant first-hand experience with. You won’t find most of the knowledge we share here in the AWS docs.

In the next lesson, we will learn about the Default Heuristic technique.

**MARK AS COMPLETED**

# The Default Heuristic

The Default Heuristic technique helps solve our struggle for searching the best AWS services and features.

**WE'LL COVER THE FOLLOWING**

* + [Optimization fallacy](https://www.educative.io/courses/good-parts-of-aws/xonxx224LRn#optimization-fallacy)
  + [The default heuristic](https://www.educative.io/courses/good-parts-of-aws/xonxx224LRn#the-default-heuristic)
  + [Our default choice](https://www.educative.io/courses/good-parts-of-aws/xonxx224LRn#our-default-choice)

**Optimization fallacy**[#](https://www.educative.io/courses/good-parts-of-aws/xonxx224LRn#optimization-fallacy)

Chasing the best tool for the job is a particularly insidious trap when it comes to making progress—especially at the beginning of a project. We consider the relentless search for the best tool to be an **optimization fallacy**—in the same category as any other premature optimization.

Searching for the optimal option is:

* Almost always expensive.
* Any belief that we can easily discover the best option by exhaustively testing each one is delusional.

To make matters worse:

* We developers tend to enjoy tinkering with new technology and figuring out how things work, and this only amplifies the vicious cycle of such a pursuit.

**The default heuristic**[#](https://www.educative.io/courses/good-parts-of-aws/xonxx224LRn#the-default-heuristic)

Instead of searching for the best option, we recommend a technique we call the default heuristic. The premise of this heuristic is that when the cost of acquiring new information is high and the consequence of deviating from a default choice is low, sticking with the default will likely be the optimal choice.

But what should your default choice be?

* It should be any option that gives you very high confidence that it will work.
* It is something you’ve used before.
* Something you understand well.
* Something that has proven itself to be a reliable way for getting things done in the space you’re operating in.
* It doesn’t necessarily have to be the theoretical best choice.
* It doesn’t have to be the most efficient. Or the latest and greatest.
* It simply needs to be a reliable option to get you to your ultimate desirable outcome.
* It should be very unlikely to fail you; you have to be confident that it’s a very safe bet. In fact, that’s the only requirement.

With this heuristic, you start making all your choices based on your defaults. You would only deviate from your defaults if you realize you absolutely have to.

Let’s take a small quiz to check your understanding of the default heuristic.

**Q**

What should your default choice be?

**A)**

It has to be a theoretical best choice.

**B)**

It has to be the most efficient.

**Correct Answer**

**C)**

It should be a reliable option to get to your ultimate desirable outcome.

**Your Answer**

**D)**

It has to be the latest or the greatest.

?????

**Oops, you got that wrong. Want to try again?**

Retake Quiz

**Our default choice**[#](https://www.educative.io/courses/good-parts-of-aws/xonxx224LRn#our-default-choice)

When you start with little experience, you might not have a default choice for everything you want to do. In this course:

|  |
| --- |
| * We’re going to share our own default choices when it comes to AWS services and features. |
| * We’re going to explain why some things became our defaults, and why other things we don’t even bother with. |

We hope this information will help you build or supplement your basket of default choices so that when you take on your next project you will be able to make choices quickly and confidently.

In the next chapter, we will look at all the good parts of AWS one by one.

# Database: DynamoDB

DynamoDB features, the pros and cons of DynamoDB indexes and when it is a great default choice for a database will be discussed in this lesson. To conclude, we will list down our recommendations on the use of DynamoDB indexes.

**WE'LL COVER THE FOLLOWING**

* + [DynamoDB vs relational database](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#dynamodb-vs-relational-database)
  + [Query processing](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#query-processing)
  + [Storage cost](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#storage-cost)
  + [Request pricing](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#request-pricing)
    - [On-demand](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#on-demand)
    - [Provisioned capacity](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#provisioned-capacity)
  + [DynamoDB indexes](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#dynamodb-indexes)
    - [Local indexes](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#local-indexes)
    - [Global indexes](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#global-indexes)

## Data structure in the cloud

Amazon describes DynamoDB as a database, but it’s best seen as a highly-durable data structure in the cloud. A partitioned B-tree data structure, to be precise.

B tree of customer data and their keys

DynamoDB is much more similar to a **Redis** than it is to a **MySQL**. But, unlike **Redis**, it is immediately consistent and highly-durable, centered around that single data structure. If you put something into DynamoDB, you’ll be able to read it back immediately and, for all practical purposes, you can assume that what you have put will never get lost.

It is true that DynamoDB can replace a relational database, but only if you think you can get away with storing all your data in a primitive B-tree. If so, then DynamoDB makes a great default choice for a database.

## DynamoDB vs relational database [#](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#dynamodb-vs-relational-database)

The following table shows some high-level differences between DynamoDB and relational database:

|  |  |
| --- | --- |
| **DynamoDB** | **Relational Database** |
| Suitable for web-scale applications, including social networks, gaming, media sharing, and Internet of Things (IoT). | Suitable for ad hoc queries; data warehousing; OLAP (online analytical processing). |
| DynamoDB is schemaless. DynamoDB can manage structured or semistructured data, including JSON documents. | The relational model requires a well-defined schema, where data is normalized into tables, rows, and columns. |
| DynamoDB requires you to do most of the data querying yourself within your application. You can either read a single value out of DynamoDB, or you can get a contiguous range of data. | Most of the data quering and processing is done close to the data side. |
| In DynamoDB, if you want to aggregate filter or sort, you have to do that yourself, after you receive the requested data range. | Data aggregation and sorting is done at the data side and the final summary is sent to application. |

## Query processing [#](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#query-processing)

Having to do most query processing on the application side isn’t just inconvenient. It also comes with performance implications. Relational databases run their queries close to the data, so if you’re trying to calculate the sum total value of orders per customer, then that rollup gets done while reading the data, and only the final summary (one row per customer) gets sent over the network. However, if you were to do this with DynamoDB, you’d have to get all the customer orders (one row per order), which involves a lot more data over the network, and then you have to do the rollup in your application, which is far away from the data. This characteristic will be one of the most important aspects of determining whether DynamoDB is a viable choice for your needs.

## Storage cost [#](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#storage-cost)

Another factor to consider is cost. As you can see in the table below, storing 1 TB in DynamoDB costs more as compared to storing 1 TB in S3.

| 1 TB in DynamoDB | 1 TB in S3 |
| --- | --- |
| $256/month | $23.55/month |

Data can also be compressed much more efficiently in S3, which could make this difference even bigger.

However, storage cost is rarely a large factor when deciding whether DynamoDB is a viable option. Instead, it’s generally ***request pricing*** that matters most.

**Q**

One of the major differences between DynamoDB and relational database is that in DynamoDB, data aggregration is done close to the data as opposed to relational database.

**Your Answer**

###### A)

True

**Correct Answer**

###### B)

False

?????

#### Oops, you got that wrong. Want to try again?

Retake Quiz

## Request pricing [#](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#request-pricing)

### On-demand [**#**](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#on-demand)

By default, you should start with DynamoDB’s on-demand pricing and only consider the provisioned capacity as cost optimization. On-demand costs $1.25 per million writes, and $0.25 per million reads. Now, since DynamoDB is such a simple data structure, it’s often not that hard to estimate how many requests you will need. You will likely be able to inspect your application and map every logical operation to a number of DynamoDB requests. For example, you might find that serving a web page will require four DynamoDB **read** requests. Therefore, if you expect to serve a million pages per day, your DynamoDB requests for that action would cost $1/day.

**Let’s do a fun exercise:** Use the below widget to calculate the on-demand cost when $0.25 is the cost per million reads. You just have to put the number of **read** requests (in millions) required for serving a web page, in the first cell. It will automatically generate the cost in the next cell.

|  |  |
| --- | --- |
| No. of read request (millions per second) | On-demand read cost ($) |
| 4 | *f*1 |

**Here’s another activity:** Use the below widget to calculate the on-demand cost when $1.25 is the cost per million writes. You just have to put the number of **write** requests required for serving a web page, in the first cell. It will automatically generate the cost in the next cell.

|  |  |  |
| --- | --- | --- |
|  | A | B |
| 1 | No. of write request (millions per second) | On-demand write cost ($) |
| 2 | 100 | *f*125 |

### Provisioned capacity [**#**](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#provisioned-capacity)

If the performance characteristics of DynamoDB are compatible with your application and the on-demand request pricing is in the ballpark of acceptability, you can consider switching to provisioned capacity. On paper, that same workload that cost $1/day to serve 1 million pages would only cost $0.14/day with provisioned capacity, which seems like a very spectacular cost reduction. However, this calculation assumes that:

* Requests are evenly distributed over the course of the day
* There is absolutely zero capacity headroom. (You would get throttled if there were a million and one requests in a day.)

Obviously, both of these assumptions are impractical. In reality, you’re going to have to provide abundant headroom in order to deal with the peak request rate, as well as to handle any general uncertainty in demand. With provisioned capacity, you will have the burden to monitor your utilization and proactively provision the necessary capacity.

**Let’s do a fun exercise:** Use the below widget to calculate the provisioned cost when $0.00013 is the cost per read capacity unit (RCU) per hour. You just have to put the number of **read** requests per hour required for serving a web page, in the first cell. It will automatically generate the cost in the next cell.

|  |  |
| --- | --- |
| No. of read requests per hour | Provisioned read cost per hour ($) |
| 50000 | *f*6.499999999999999 |

**Here’s another activity:** Use the below widget to calculate the provisioned cost when $0.00065 is the cost per write capacity unit (WCU) per hour. You just have to put the number of **write** requests per hour required for serving a web page, in the first cell. It will automatically generate the cost in the next cell.

|  |  |
| --- | --- |
| No. of write requests per hour | Provisioned write cost per hour ($) |
| 50000 | *f*32.5 |

In general:

* You will almost always want to start with on-demand pricing (no capacity management burden).
* Then, if your usage grows significantly, you will almost always want to consider moving to provisioned capacity (significant cost savings).

However, if you believe that on-demand pricing is too expensive, then DynamoDB will very likely be too expensive, even with provisioned capacity. In that case, you may want to consider a relational database, which will have very different cost characteristics than DynamoDB.

It is important to note that:

* With on-demand pricing, the capacity you get is not perfectly on-demand. Behind the scenes, DynamoDB adjusts a limit on the number of reads and writes per second, and these limits change based on your usage. However, this is an opaque process and, if you want to ensure that you reserve capacity for big fluctuations in usage, you may want to consider using provisioned capacity for peace of mind.

**NOTE:** It is very important to note that the prices mentioned above are only for the purpose of understanding cost comparisons. These prices are subject to change anytime by AWS. Therefore, most current prices should be referenced for final business decisions.

**Q**

With on-demand request pricing, you will have the burden to monitor your utilization and proactively provision the necessary capacity.

###### A)

True

Explanation

With provisioned capacity, you have the burden to monitor your utilization and proactively provision the necessary capacity.

**Correct Answer**

###### B)

False

----

#### You skipped the question. Would you like to try again?

Retake Quiz

## DynamoDB indexes [#](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#dynamodb-indexes)

A final word about DynamoDB indexes. They come in two flavors:

* Local indexes
* Global indexes

Local indexes came first in early 2013, and global indexes were added just a few months later.

### Local indexes [**#**](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#local-indexes)

The only advantage of local indexes is that they’re immediately consistent, but they do come with a very insidious downside. Once you create a local index on a table, the property that allows a table to keep growing indefinitely goes away. Local indexes come with the constraint that all the records that share the same partition key need to fit in 10 GB, and once that allocation gets exhausted, all write operations with that partition key will start failing.

### Global indexes [**#**](https://www.educative.io/courses/good-parts-of-aws/xl8QO9xKvxE#global-indexes)

On the other hand, global indexes don’t constrain your table size in any way, but reading from them is eventually consistent (although the delay is almost always unnoticeable). Global indexes also have one insidious downside, but for most scenarios, it is much less worrisome than that of local indexes.

DynamoDB has an internal queue-like system between the main table and the global index, and this queue has a fixed (but opaque) size. Therefore, if the provisioned throughput of a global index happens to be insufficient to keep up with updates on the main table, then that queue can get full. When that happens, disaster strikes:

* All write operations on the main table start failing.

The most problematic part of this behavior is that there’s no way to monitor the state of this internal queue. So, the only way to prevent it is:

* To monitor the throttled request count on all your global indexes, and then to react quickly to any throttling by provisioning additional capacity on the affected indexes.

Nevertheless, this situation tends to only happen with highly active tables, and short bursts of throttling rarely cause this problem.

# ↓ OUR RECOMMENDATIONS ↓

|  |
| --- |
| * Unless you know for sure that you won’t ever exceed the storage limit of 10GB, we recommend avoiding **local indexes**. |
| * **Global indexes** are still very useful, but keep in mind the fact that they’re eventually consistent and that they can indirectly affect the main table in a very consequential manner if they happen to be underprovisioned. |

In the next lesson, we will take a look at when should we use S3.



