Unit 3 Tutorials: Computer Engineering Concepts

INSIDE UNIT 3

Definition of an Algorithm

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the definition of an algorithm by discussing:

1. DEFINITION OF ALGORITHM
2. ALGORITHM EXAMPLE

In this unit, we'll look briefly into some intermediate and advanced level concepts that pertain to the Web Developer career. This section is intended to expose you to things that you would otherwise not encounter if you hadn't learned the basics of web development. We won't go deep here- we just want to give you a sense of what it's like to be a web developer once you have mastered the basics of the syntax of a language and the technologies of the web. You could say that this is where the artistry of the career starts to come into play. Beyond technological understanding, being a web developer is really creative and design-oriented. The challenges in this unit will uncover why.

1. DEFINITION OF ALGORITHM

It's hard to read about technology in the news without hearing the term**algorithm**. So, it's likely that most people, even those not working in technology, have a working definition of the term. This is a useful entry point.

Algorithms, at their highest level, are used for calculating or generating something that would normally take a lot of time and resources to figure out. An algorithm in it's broadest sense is a repeatable set of instructions to follow that, when given an input or a set of inputs, can produce some useful output or set of outputs. An algorithm is like a recipe for calculating something that's otherwise complex or difficult to produce.

Algorithms aren't just technological- they can be completed by humans too. A cooking recipe is, in itself, an algorithm. There are inputs—ingredients, a set of instructions—how to mix, how long to bake something and at what temperature, and outputs—the delicious food to eat when you're finished. But algorithms tend to deal with intellectual matter rather than solid matter.

We see algorithms for:

- Recommending things based on a user's preferences (think of Netflix) *Mapping based on different transportation modes and preferred road types (think of GPS aps)
- Searching for information based on what Is likely to be most relevant (think of Googling something)
- Categorizing or identifying an unknown item based on its visual appearance (think of bird watching or star gazing apps)
- Predicting future behavior based on past behavior (think of the recommendations that you get on Amazon); and more!

There really is no limit to what an algorithm can attempt to do. What really matters, though, is how well an algorithm does what it sets out to do.
Algorithms have become technological because when we make complex processes into a repeatable set of steps, it's possible to have a computer complete those steps.

This is exciting because it means that the creation of sophisticated algorithms can automate what would otherwise take a lot of time and effort for a human to complete. Many companies invest in the development of algorithms because there is value in providing automatic steps for complex processes.

TERMS TO KNOW

Algorithm
Algorithms are used for calculating or generating something that would normally take a lot of time and resources to figure out.

---

2. ALGORITHM EXAMPLE

In order to understand the way that algorithms are implemented using code, we'll look at the example of the recommendation algorithm- the ability to take a user's history and predict what products or content they would like to buy or see in the future.

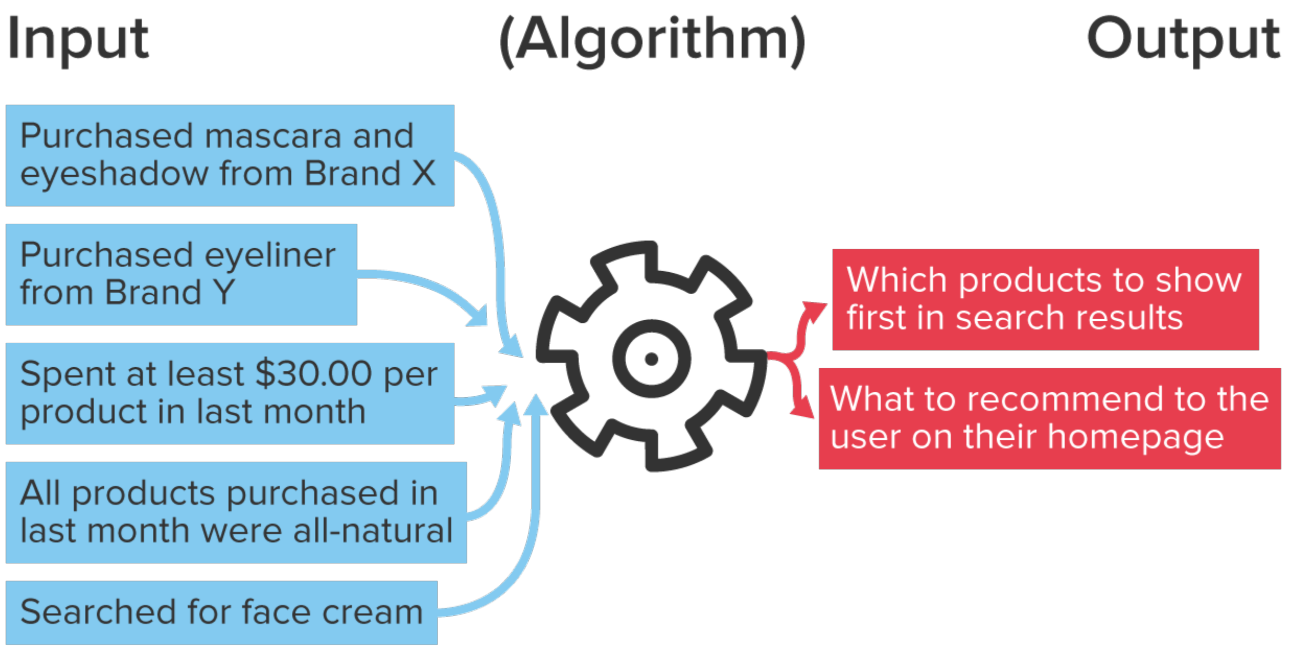
The utility of recommendation algorithms is clear—almost any e-commerce or streaming platform will benefit greatly from being able to steer their users' toward new products and content that they will enjoy.

In this example, we'll imagine an e-commerce platform where people purchase makeup and skincare. The goal is to utilize a user's past shopping history to make engaging recommendations on a user's homepage for products that they might like to purchase next.

The input for the algorithm is the user profile that we are trying to recommend products for and the products that the user has purchased in the past.

The output for the algorithm is the set of the top five products to recommend to the user.

It's worth noting here that the output for the algorithm is dynamic- it's changing perhaps all the time, and is different from user to user. If it were the same all the time, or the same from every user, we might not require a computational algorithm to provide a recommendation.


Input
(Algorithm)
Output
Purchased mascara and eyeshadow from Brand X
Purchased eyeliner from Brand Y
Spent at least $30.00 per product in last month
All products purchased in last month were all-natural
Searched for face cream
Which products to show first in search results
What to recommend to the user on their homepage

Input and Output from an Algorithm.

These are the steps for two possible algorithms for recommending things based on a user's preference.

Algorithm Simple (A basic algorithm)

STEP BY STEP

1. Look up the last product that the user purchased.
   - If the user hasn't purchased anything before, choose the top 5 most popular products in the entire store.
   - If the user has purchased something, go a different route.

2. Choose four products from the same brand of the last product that they purchased that are closest

in price to the last item they purchased.
3. Choose 1 product that is the same type of product (i.e. mascara, moisturizer) as the last product they purchased.

Algorithm Complex (More steps, more accurate)

STEP BY STEP

1. Look up the last product that the user purchased.
   - If the user hasn't purchased anything before, choose the top 5 most popular products in the entire store.
   - f the user has purchased even one item, evaluate the user's entire purchase history in relation to other users.

2. Reviewing all other users' history, compare how similar others' purchase history is to the user in question.
   - A user with identical history will have a comparison estimation of 1.0. A user with completely different history will have a comparison estimation of -1.0. We will compute these coefficients using the Pearson correlation formula.
   - Once we have the correlation scores for all other users against our user in question, we will find the products that users most similar to the user in question bought.

3. Starting with the most similar user (assuming that user has bought at least 5 products) we'll recommend the top 5 most recent products that the user purchased.
4. If the top user hasn't purchased 5 products, we'll keep going down the list from most similar to least similar until we have 5 products.

After walking through the steps to create and use this recommendation algorithm, next time you are ordering something online, think about how many algorithms could be involved in each part of the ordering process.

TERMS TO KNOW

Algorithm
Algorithms are used for calculating or generating something that would normally take a lot of time and resources to figure out.

Comparing Algorithms

by Devmountain Tutorials

WHAT'S COVERED

This section will explore how to compare and plan algorithms by discussing:

1. COMPARING ALGORITHMS
2. PSEUDOCODE

1. COMPARING ALGORITHMS

Keep in mind, there are many possible algorithms that can solve one problem. As we can see from the previous section, at least two (but in reality, many more) algorithms exist that can choose the top 5 products to recommend to a user on an e-commerce website. It's worth noting that writing an algorithm is a skill in and of itself. Creating a unique algorithm is something that a web developer might be asked to do.

However, what's much more likely is that a web developer would need to choose from a set of existing algorithms for a particular task, and implement one of them. Many known algorithms have already been created for the various use cases we've mentioned in this unit. To get a sense of this task, let's try to compare and contrast the two algorithms we saw in the previous section.

	Algorithm Simple	Algorithm Complex
Number of Calculations	+	+++
Accuracy	++	+++
Easy to Understand	+++	+

The chart above estimates the way that the two algorithms compare. The categories of number of calculations, accuracy, and ease of understanding are some of the common drivers for the decision about which algorithm to choose for a certain task.

With some tasks, such as mapping a route to get from one place to another, accuracy is of the highest importance. You wouldn't want to end up 1 mile west of where a birthday party is occurring, right?

With our task, accuracy isn't as important. While we want to show recommendations that are engaging to our shoppers, the ability to predict what they will want to buy next isn't necessarily going to make or break our e-commerce platform.

Another thing this chart showcases is that there is a tradeoff with different categories. Specifically, and the number of calculations, or "runtime complexity" increases, so does the difficulty of quickly understanding an algorithm.

It might be in a web developer's best interest to choose an algorithm that's easy for the rest of the team to understand, especially if accuracy isn't the top concern. Algorithms that are easier to understand translate to code that is more likely to stay bug-free in the long run.

Other times, what we gain in accuracy through a more complex algorithm isn't necessarily worth it. For our very simple algorithm, we get a fairly accurate set of outputs with Algorithm Complex. Even though there aren't a ton of calculations, we get a pretty good set of product recommendations. The cost of upping the accuracy level by just a little bit through using a lot of calculations just isn't quite worth it in this case.

---

2. Pseudocode

Continuing with our study of the two algorithms for e-commerce, let's talk about an important part of any coders' languages- pseudocode! Pseudocode is both exactly what it sounds like, but also much more important than it sounds. Pseudocode is a way of notating what a set of computer code will do, step by step, section by section, but without utilizing the actual rules and grammar of a particular programming language.

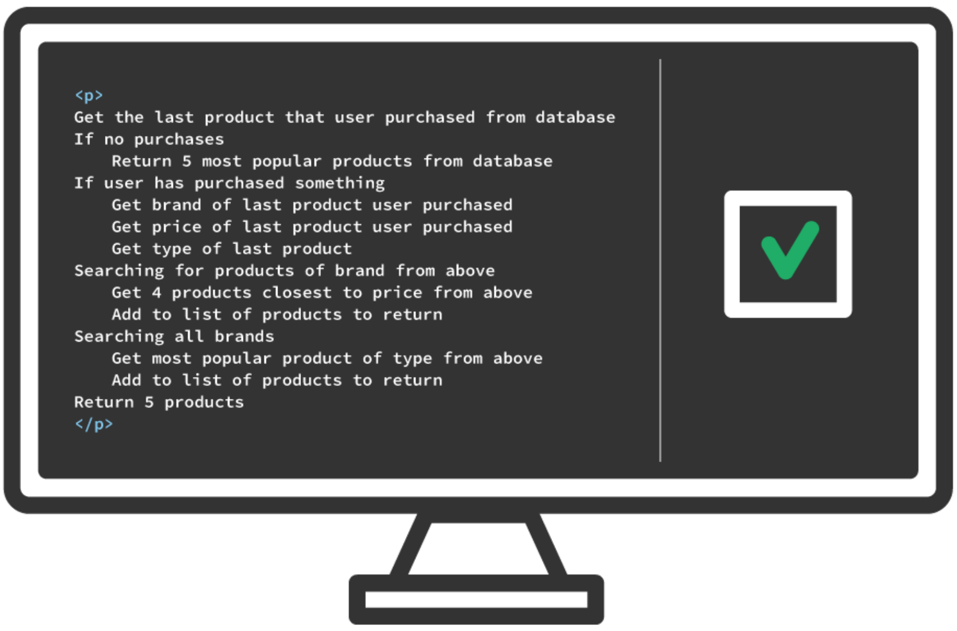
At first glance, this might seem like a tool to simply allow many programmers who don't speak the same language communicate. While it can serve that purpose, pseudocode actually provides an even broader application.

Pseudocode allows algorithms to be documented in a *codeful*, yet language agnostic manner. Pseudocode is codeful in that the level of detail within pseudocode is fairly similar, line for line, with what you would see in a real codebase.

Pseudocode is also language agnostic; it doesn't require the knowledge of any one programming language in order to understand it. In fact, a majority of pseudocode doesn't even require that the reader knows how to program at all!

The other advantage that pseudocode provides is an important planning step in between understanding what the code should do and actually writing code.

Pseudocode for Algorithm A



```
<p>
Get the last product that user purchased from database
If no purchases
    Return 5 most popular products from database
If user has purchased something
    Get brand of last product user purchased
    Get price of last product user purchased
    Get type of last product
Searching for products of brand from above
    Get 4 products closest to price from above
    Add to list of products to return
Searching all brands
    Get most popular product of type from above
    Add to list of products to return
Return 5 products
</p>
```

Similar to writing effectively in English, there are a lot of important choices to make in order to write code in a clear, concise, and efficient manner. Pseudocode serves the purpose of a rough draft or outline for the actual code in a project.

TERM TO KNOW

Pseudocode
Pseudocode is a way of notating what a set of computer code will do, step by step, section by section, but without utilizing the actual rules and grammar of a particular programming language.

TERMS TO KNOW

Pseudocode
Pseudocode is a way of notating what a set of computer code will do, step by step, section by section, but without utilizing the actual rules and grammar of a particular programming language.

Definition of API

by Devmountain Tutorials

WHAT'S COVERED

This section will explore Application Programming Interface (API) by discussing:

1. DEFINITION OF API
2. API EXAMPLE

1. DEFINITION OF API

As we saw in the last challenge, an algorithm collects different sets of information, also known as data, and puts it together to make a decision for the user.

Think about the somewhat obsolete work of a travel agent. She begins with what she knows about her clients' travel preferences – where they would like to go, what they like to do, and their preferred mode of travel.

Thus begins her travel algorithm: she pulls information from the best airline companies and tour companies. She selects flights and tourist activities for her clients. Depending on airlines' availability, she makes necessary adjustments to tourist activities. Depending on her travel agency's partnerships, she applies available discounts to airlines and activities. All of this work encompasses the algorithm of the travel agent. It's complex work that can be done in a variety of ways.

Nowadays, we are our own travel agents. We don't need specialized agents to call airline companies or tell us about the available activities for a certain city we've never been to. That data is available to us directly through the internet.

We are able to interact with the raw source of the information to make our own complex set of decisions. In this analogy, the interface that we humans use to get data about airline flights is not unlike an **API**, or an **Application Programming Interface**.

As we know, algorithms are now being completed by computers, and computers need a way to talk to their own data sources. Computers need to be able to pull in different sets of data just like a travel agent would back in the old days.

f you recall from Units 1 and 2, API stands for Application Programming Interface. But, the acronym doesn't really provide enough information to understand what an API means to a web developer. An API is the set of options a computer (or code) has to be able to use to talk to an external data source. In this challenge, we'll get to explore API's and their importance a little more.

2. API EXAMPLE

The API for an airline flight data source would be composed of a set of**methods**, or functions, that are each a unique question that the computer code can “ask” the data source.

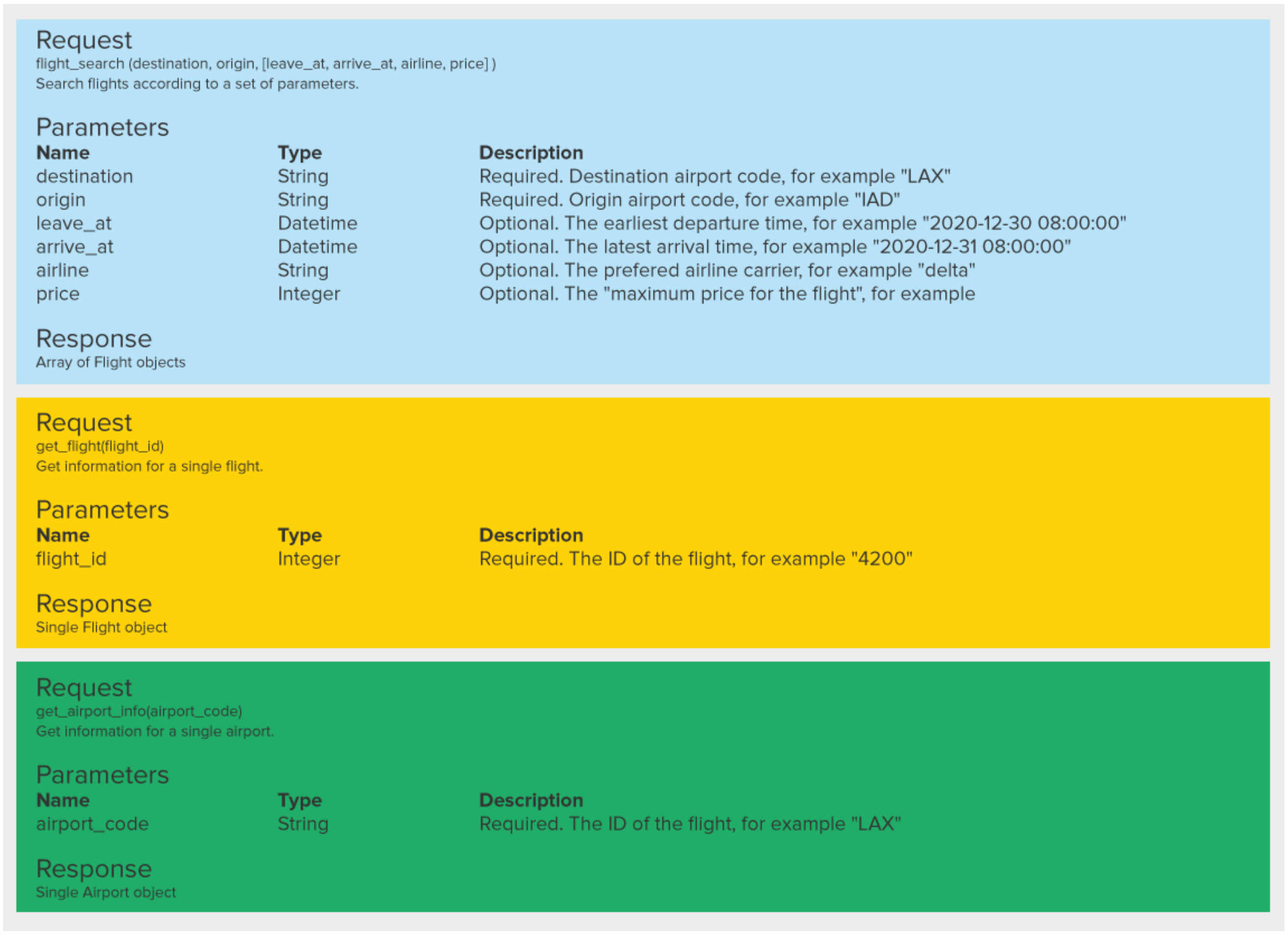
For example, the code might execute the method **flight_search** to ask the question “Can I have a list of all direct flights from San Francisco to Lisbon that leave on February 20, 2021?” and the data source would respond with that list.

It’s important to note that API refers to the interface for the data—not the data itself. When we think about APIs, we are thinking about the set of questions, or methods, that we are able to use to speak to a dataset.

As programmers, we might not have direct access to a database with the data that we want. The reasons for this are simple—we don’t maintain the database of flight information. Instead, there is some external entity that has that responsibility. If we did have direct access to the flight database, we could query it using **SQL**, the language of databases.

Instead, the programmers who maintain the flight data have exposed their data to external programmers who may want to use it via an API. Below, you can see an example of the API for the flight data. We use the API by installing a code package and then writing code according to the API’s documentation.

Here is an example of what the documentation might look like for the flight data API.



Request
flight_search (destination, origin, [leave_at, arrive_at, airline, price])
Search flights according to a set of parameters.

Parameters

Name	Type	Description
destination	String	Required. Destination airport code, for example "LAX"
origin	String	Required. Origin airport code, for example "IAD"
leave_at	Datetime	Optional. The earliest departure time, for example "2020-12-30 08:00:00"
arrive_at	Datetime	Optional. The latest arrival time, for example "2020-12-31 08:00:00"
airline	String	Optional. The prefered airline carrier, for example "delta"
price	Integer	Optional. The "maximum price for the flight", for example

Response
Array of Flight objects

Request
get_flight(flight_id)
Get information for a single flight.

Parameters

Name	Type	Description
flight_id	Integer	Required. The ID of the flight, for example "4200"

Response
Single Flight object

Request
get_airport_info(airport_code)
Get information for a single airport.

Parameters

Name	Type	Description
airport_code	String	Required. The ID of the flight, for example "LAX"

Response
Single Airport object

Sample API: Flight Data

As you can see, there are three methods in this excerpt of API documentation. Think of each method as a question that we can ask the database.

Methods can take parameters, or additional pieces of data that change the question we’re asking. Some parameters are required in order to use the method, while some are optional and can be provided to narrow the set of data that you're asking for.

TERMS TO KNOW

Interface
A way for two systems to interact.

Method
A question that we can ask the database.

SQL
The language of databases.

TERMS TO KNOW

Interface
A way for two systems to interact.

Method
A question that we can ask the database.

SQL
The language of databases.

The Importance of Language

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the importance of language in creating APIs by discussing:

1. WEB-BASED APIs
2. DESIGN OF AN API

1. WEB-BASED APIs

The API example we've been looking at is a specific example that lets us see how useful APIs can be. While web-based APIs are increasingly common, especially as different apps and companies need to integrate with the surrounding digital world, APIs aren't limited to just the purpose of pulling in data.

APIs are everywhere in web development. They provide the interface for programmers to interact with all the different types of predefined objects within a particular framework. As you get more experienced as a Web Developer, your knowledge of various languages, frameworks, and external APIs expands gradually.

2. DESIGN OF AN API

The design of an API is one of the most creatively challenging tasks that a developer may encounter in their career. Designing an API requires a developer to think about the other programmers who will use the API, and how to make the API easy to use for their use cases.

APIs that are well-designed showcase thoughtful and purposeful language decisions in the naming of methods and parameters. The questions and objects that a user can interact with should feel intuitive and self-evident.

While the API that we looked at deals with real-life objects like flights and airports, it's more common for APIs to deal with virtual objects, the definitions of which are also up to the API designer.

APIs can be vast, requiring pages and pages of documentation. In the case of larger APIs, the language choice and scope of the methods can be crucial for the API to be intuitive to use.

In this challenge, we looked at the importance of API's for two systems to interact and share data, or information. In the final challenge, we'll take a look at databases and the purpose they serve.

Definition of Database

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the purpose of databases by discussing:

1. DEFINITION OF DATABASE

DEFINITION OF DATABASE

A **database** is the rawest form that data can take in a web application, and it's often referred to as the "source of truth". Without a database, a web developer has no means of programming user sessions that refer back to one another, such as when a user logs in and performs a set of actions that they expect to be accessible next time they login. From saving basic user data such as usernames and emails to saving more complex data such as documents and product purchases, a database enables a wide variety of dynamic functionality.

Before a web developer can store and retrieve data from a database, the database needs to be designed and created. This task is yet another case where coding takes on a more creative, design-oriented quality. The design of a database is not only critical to any applications' future functionality, but it also is difficult to modify once it's been designed, so it has lasting consequences for potentially years to come.

Thankfully, the basic building materials for a database are relatively few.

You have **tables**, which are like buckets that contain sets of uniform items. Items in a table are called **records**—each record is can be likened to a row in a spreadsheet. Records have**fields**, which are attributes of the record—not unlike the column in a spreadsheet. Lastly, tables can have **relationships** to other tables.

So, in order to complete a basic database design, a web developer would need to specify which tables need to exist, and within each table, which fields will pertain to the records in each table. Lastly, the developer would need to decide, or determine, what the relationships between the tables should be.

TERMS TO KNOW

Database
Resource that saves basic user data such as usernames and emails, as well as saving more complex data such as documents and product purchases. A database enables a wide variety of dynamic functionality.

Table
Data collection objects that are like buckets that contain sets of uniform items, or records.

Record
An item in a table that can be likened to a row in a spreadsheet, including attributes of the record called fields.

Field

Field

Attributes of the record—not unlike the column in a spreadsheet.

Relationships

Developers determine the connections between tables and how they relate to each other.

TERMS TO KNOW

Database

Resource that saves basic user data such as usernames and emails, as well as saving more complex data such as documents and product purchases. A database enables a wide variety of dynamic functionality.

Field

Attributes of the record—not unlike the column in a spreadsheet.

Record

An item in a table that can be likened to a row in a spreadsheet, including attributes of the record called fields.

Relationships

Developers determine the connections between tables and how they relate to each other.

Table

Data collection objects that are like buckets that contain sets of uniform items, or records.

Design of a Database

by Devmountain Tutorials

WHAT'S COVERED

This section will explore the design of databases by discussing:

1. TABLES
2. RELATIONSHIPS

Let’s try designing a database for a photo sharing application. The basic functionality for this application includes users that make posts, showcasing one or many photos per post. Other users can add comments to each post. Believe it or not, just those two sentences of functionality will result in a moderately complex database.

1. TABLES

As was mentioned in the last challenge, there should be a table for each “kind” of record in our database. Tables contain uniform data—so the things that go in each table should all be similar.

Determining the first few tables is usually fairly simple. We need a table for each entity, or object, that was mentioned in the description of the application’s functionality including users, posts, photos, and comments. We’ll need the following tables for each category of users, posts, photos, comments.

Let’s talk about the records we might find in each table.

TABLE 1: USERS

The users table will contain a row, or record, for each user in our application. When someone new registers to user our app, we’ll add a new record in this table. Here are the columns, or fields, that will be present in our users table. We might want to add more fields later, but here’s where we’ll start.

- **User ID**: A unique numeric identifier for each user.
- **Email**: User's email address.
- **Password Hash**: Aversion of the user's password that we can use to verify the user can login to their account.
- **Username**: Name to be associated with the user's account.

TABLE 2: POSTS

This table might seem a little boring at first, but we’ll need it later.

- **Post ID**: A unique numeric identifier for each post.
- **Description**: The text that the user will associate with the post.

TABLE 3: PHOTOS

Instead of storing photo information in the posts table, we will create a separate table to store photos. This lets us have a separate record for each photo, rather than having to stuff all the photo information into the post table—too messy!

- **Photo ID**: A unique numeric identifier for the photo.
- **Location**: The place where the photo is stored on our computer.

TABLE 4: COMMENTS

Similar to the photos and posts situation, we don’t want to store a photo’s comments in the photos table. That would result in a bloated, complicated mess in our photos table. Instead, we’ll have a separate record for each comment, all stored in the comments table. Here are the fields for the comments table.

- **Comment ID**: A unique numeric identifier for the comment.
- **Comment Text**: The text of the comment.

Now that there are buckets, or tables, for the data in this photo sharing application, we are able to store data when users interact with our application.

For example, when a user registers, a user record can be added to the user table. When a user adds a post, we are now able to add a post record to the post table, as well as one or several photo records to the photo table. When another user comments on a post, we have a place to store comment data for each comment.

2. RELATIONSHIPS

You may have noticed a problem with the tables and fields we’ve created. According to the fields in our various tables, there is currently no way to determine which post belongs to which user, which photos belong to which post, which comment was made about which post, nor which comment was made by which user.

In order to store this kind of data, we need to find a way to acknowledge and store the relationships between the tables.

Relationship 1: Each post belongs to a single user
Relationship 2: Each photo belongs to a single post
Relationship 3: Each comment was made by a single user
Relationship 4: Each comment is about a single post

These relationships can also be represented like this:

A user has many (at most) posts

A post has many (at most) photos

A post has many (at most) comments

A user has many (at most) comments

Or like this:

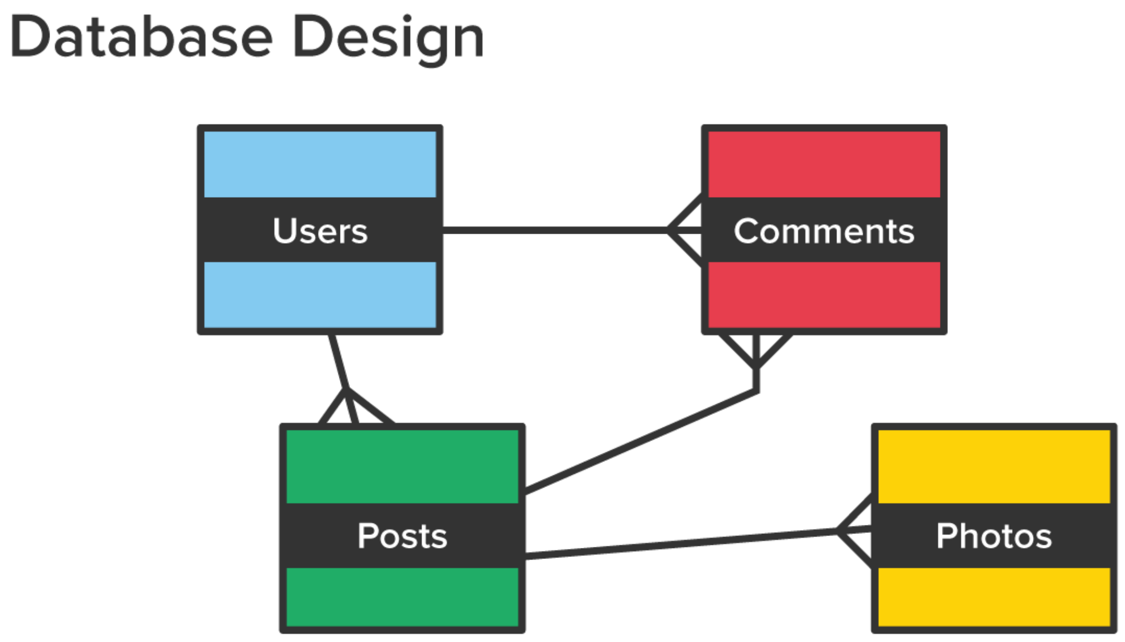


Diagram of the tables and relationships in a databse.

We can store data about the relationships between tables by adding fields to our tables that refer to another record in another table. For example—to represent the relationship that "each post belongs to a single user" we can do the following:

1. Leave the users table as is.
2. Add a field to the post table called User ID. This field will be a place to store the User ID for the user who created a given post!

USERS TABLE

User ID	Email	Password Hash	Username

POSTS TABLE

Post ID	Description	User ID

We'll replicate this for the post/photo relationship as well.

POSTS TABLE

Post ID	Description	User ID

PHOTOS TABLE

Photo ID	Description	Post ID

This process of fields referring to fields in other tables allows the tables to relate data across the database, while still keeping “like” data together, in its own separate table. All photo records can exist in a table where fields are solely about photo data, but we can refer to the Post ID field in the photo table to understand which post a given photo belongs to.

Database design is a highly complex task that takes a lot of planning and evaluating. Determining the right tables, fields, and relationships is not as straightforward that one might assume, but once data is represented correctly within a database, a web application has a solid foundation to build upon.

Hopefully, you’ve gotten a feel for the kind of critical thinking that’s needed to design a database well. It’s one of the many tasks for web developers that requires very little coding, and instead, careful thought, planning, and collaboration.

Terms to Know

Algorithm

Algorithms are used for calculating or generating something that would normally take a lot of time and resources to figure out.

Database

Resource that saves basic user data such as usernames and emails, as well as saving more complex data such as documents and product purchases. A database enables a wide variety of dynamic functionality.

Field

Attributes of the record—not unlike the column in a spreadsheet.

Interface

A way for two systems to interact.

Method

A question that we can ask the database.

Pseudocode

Pseudocode is a way of notating what a set of computer code will do, step by step, section by section, but without utilizing the actual rules and grammar of a particular programming language.

Record

An item in a table that can be likened to a row in a spreadsheet, including attributes of the record called fields.

Relationships

Developers determine the connections between tables and how they relate to each other.

SQL

The language of databases.

Table

Data collection objects that are like buckets that contain sets of uniform items, or records.