MySQL for the Internet of Things

Data management for sensors and connected devices

Charles Bell

Apress®

MySQL for the Internet of Things

Charles Bell

Apress®

MySQL for the Internet of Things

I dedicate this book to my major professors Drs. James E. Ames IV and Lorraine M. Parker whose tutelage both academic and personal have instilled in me a love for the art and science of database systems.

Dr. Charles Bell

Contents at a Glance

Contents

About the Author

Dr. Charles Bell conducts research in emerging technologies. He is a member of the Oracle MySQL Development team as a development manager directing development of MySQL high availability solutions. He lives in a small town in rural Virginia with his loving wife. He received his doctorate of philosophy in engineering from Virginia Commonwealth University in 2005. His research interests include database systems, software engineering, sensor networks, and 3D printing. He spends his limited free time as a practicing Maker focusing on microcontroller and 3D printers and printing projects.

About the Technical Reviewer

Peter Adams has been crafting web and mobile applications for businesses of every size since 1998. To make a living, he manages a small development team working with businesses to build their technology stack. To live when he's not behind the keyboard, he takes advantage of the Arizona climate to hike, mountain bike, and motorcycle year round. Occasionally, he takes time to write about his adventures at `www.mymegaverse.org/blog`. More often, you can find him keeping his clients updated at `www.pingdevelopment.com`.

Acknowledgments

I would like to thank all of the many talented and energetic professionals at Apress. I appreciate the understanding and patience of my editor, Jonathan Gennick, and managing editor, Jill Balzano. They were instrumental in the success of this project. I would also like to thank the army of publishing professionals at Apress for making me look so good in print. Thank you all very much!

I'd like to especially thank the technical reviewer, Peter Adams, for his often-profound insights, constructive criticism, and encouragement. I'd also like to thank my friends for their encouragement and suggestions for things to include in the book.

Most importantly, I want to thank my wife, Annette, for her unending patience and understanding while I spent so much time with my laptop.

Introduction

Internet of Things (IoT) solutions are not nearly as complicated as the name may seem to indicate. Indeed, the IoT is largely another name for what we have already been doing. You may have heard of "connected devices" or "Internet-ready" or even "cloud-enabled." All of these refer to the same thing—be it a single device such as a toaster or a plant monitor or a complex, multidevice product like home automation solutions. They all share one thing in common: they can be accessed via the Internet to either display data or interact with the devices directly. The trick is applying knowledge of technologies to leverage them to the best advantages for your IoT solution. In this book, we explore how to leverage MySQL in your IoT solution as a means to store your data.

Intended Audience

I wrote this book to share my passion for MySQL and IoT solutions. I especially wanted to show how anyone can use MySQL to store IoT data—even from small microcontrollers. Indeed, I want to share with the world the power and ease of installation and use of MySQL, the world's most popular open source database.

The intended audience therefore includes anyone interested in learning how to install and use MySQL, hobbyists and enthusiasts who need a way to store data in their IoT solutions, and designers and engineers building commercial IoT solutions.

How This Book Is Structured

The book was written to guide the reader from a general knowledge of MySQL to expertise in developing MySQL solutions for the IoT. The first several chapters cover general topics including a short introduction to the Internet of Things, how data is generated and stored, and some of the available hardware for IoT. Additional chapters present a primer on MySQL including how to install and configure database servers. Throughout the book are examples of how to implement many of the concepts presented. The following is a brief overview of each chapter included in this book.

- **Chapter 1, "The Internet of Things and Data":** This chapter presents and answers the questions of what the IOT is and how IOT solutions are constructed. You are introduced to some terminology to describe the architecture of IOT solutions as well as some examples of well-known IOT solutions. The chapter concludes with a discussion of the two most critical issues of IOT solutions through practical examples: big data and security.

- **Chapter 2, "Hardware for IOT Solutions":** This chapter discusses some of the hardware available for building IoT solutions including the Arduino line of microcontroller boards, low-cost (low-power) computing boards such as the Raspberry Pi, and the various communication hardware you can use to connect the nodes from using an Ethernet or WiFi network connecting to the Internet to low-cost, low-power wireless communication modules (XBee radios) for connecting sensor nodes. The chapter also presents a tutorial of the Arduino programming environment along with examples of how to use the Arduino with sensors. The chapter concludes with a brief look at the types of sensors available for building IoT solutions.

- **Chapter 3, "How IOT Data Is Stored":** This chapter examines some of the methods available to you for storing data. You will see examples of how to read and write data in files on the Arduino and Raspberry Pi. The chapter also presents the benefits, considerations, and recommendations for deploying a database server in your IOT solution. The chapter concludes with a list of the best practices for designing a network of nodes for your IOT solution.

- **Chapter 4, "Data Transformation":** This chapter presents practical questions to ask when considering the data as well as several examples of annotation and aggregation in both Arduino and Python code. This chapter also includes considerations for storing IOT data in a database such as implementing annotations and aggregations in the database rather than in code.

- **Chapter 5, "MySQL Primer":** This chapter dives into discovering the power of using a database server, learning how database servers store data, and seeing how to issue commands for creating databases and tables for storing data as well as commands for retrieving that data. The chapter is a primer on MySQL including how to get started with your own IOT data through database and code examples.

- **Chapter 6, "Building Low-Cost MySQL Data Nodes":** This chapter presents a crash course on how to set up a Raspberry Pi, install MySQL, and use them. The chapter also includes examples on how to write data to a MySQL database server using an Arduino sketch and a Python program on another machine (Raspberry Pi, BeagleBone Black, pcDuino, and so on). The chapter concludes with a brief look at the high availability features of MySQL.

- **Chapter 7, "High Availability IOT Solutions":** This chapter discusses what high availability is and how high availability concepts can be realized. You will also learn key high availability concepts, tools, and techniques for MySQL including backup and recovery and replication. You also will discover how to implement fault tolerance for collecting data on microcontrollers.

- **Chapter 8, "Demonstration of High Availability Techniques":** This chapter presents more about MySQL replication including some helpful tips and techniques for setting up and using replication in your IOT solutions. The chapter also presents some examples of high availability concepts including a simple round-robin read scale-out solution for Python, hardware for building a redundant data collector with failover, and how to build in fault tolerance for data collectors so that you don't lose data should the communication pathway from data collector to database fail.

How to Use This Book

This book is designed to guide you through learning more about what the Internet of Things is, discovering the power of MySQL and database systems, and seeing how to build your IoT solutions so that they are more reliable by leveraging high availability techniques.

If you are familiar with some of the topics early in the book, I recommend you skim them so that you are familiar with the context presented so that the later chapters, especially the examples, are easy to understand and implement on your own. You may also want to read some of the chapters out of order so that you can get your project moving, but I recommend going back to the chapters you skip to ensure you get all of the data presented.

If you are just getting started with MySQL and your IoT solutions or are not well versed in database systems, I recommend reading the book in its entirety before developing your own IoT solution. That said, many of the examples permit you to build small examples that you can use to help learn the concepts. For example, you can learn how to read sensors, store data, build a data aggregator, and more, for sensor networks and IoT solutions.

Downloading the Code

The code for the examples shown in this book is available on the Apress web site, `www.apress.com`. You can find a link on the book's information page on the Source Code/Downloads tab. This tab is located in the Related Titles section of the page.

Contacting the Author

Should you have any questions or comments—or even spot a mistake you think I should know about—you can contact the author at `drcharlesbell@gmail.com`.

■ ■ ■

The Internet of Things and Data

The Internet has enabled developers to create solutions that produce data that can be viewed by anyone anywhere in the world. Adapting prototypes or smaller versions of a solution to incorporate the Internet can be a challenge. It is not as simple as taking a working solution on a local network or similar communication mechanism and adding Internet connectivity. For example, growing your sensor network from a few sensors monitoring data viewed by a few people to a sensor network incorporating hundreds of sensors with the data viewable by potentially everyone may require redesigning your communication methods, data collection, and data storage.

Not only do you have to figure out how to scale the communication among your sensors, data collectors, and data-hosting services or database servers, you also have to deal with an explosion of data. That is, capturing data from a dozen sensors is relatively easy and may not require much in the way of careful planning to save the data, but capturing data from hundreds or even thousands of sensors is much more difficult because the data accumulates exponentially.

Clearly, storing the data on removable media or in a file is out of the question—especially so if you consider how the data will be used. Furthermore, making sense of all that data is complicated by how the data is stored or more appropriately retrieved. For example, what would happen if every device in your home, car, office, and so on, were to produce data? Add in the rising interest in wearable sensors and similar devices and you've got the potential to generate more data than any human can manage or even decipher.

However, it isn't just sensor networks that face a similar data crisis. Indeed, the greatest concern of innovators in the emerging and developing world of the Internet of Things (IOT) is the potential for a data explosion as more and more devices generate, communicate, and present data. What we need is a way to explore and exploit this data, and one place to start is how the data is gathered and stored on a smaller scale like a sensor network.

WHAT IS THE INTERNET OF THINGS?

The essence of the IOT is simply interconnected devices that generate and exchange data from observations, facts, and other data, making it available to anyone.[1] While there seems to be some marketing efforts attempting to make anything connected to the Internet an IOT solution or device (not unlike the shameless labeling of everything as "cloud"), IOT solutions are designed to make our knowledge of the world around us more timely and relevant by making it possible to get data about anything from anywhere at any time. Regardless, it is clear there is potential for the number of IOT devices to exceed the human population of the planet.

---

[1] https://en.wikipedia.org/wiki/Internet_of_Things.

This book will be your guide to successfully plan and implement the data storage component for your sensor network and similar IOT solution. Let's begin with a short discussion about IOT solutions and then look at the key challenges for developing IOT solutions focusing on the data.

IOT Solutions

An IOT solution is simply a set of devices designed to produce, consume, or present data about some event or series of events or observations. This can include devices that generate data such as a sensor, devices that combine data to deduce something, devices or services designed to tabulate and store the data, and devices or systems designed to present the data. Any or all of these may be connected to the Internet.

IOT solutions may include one or all of these qualities whether they are combined in a single device such as a web camera, use a sensor package and monitoring unit such as a weather station, or use a complex system of dedicated sensors, aggregators, data storage, and presentation such as a complete home automation system. Figure 1-1 shows a futuristic picture of all devices everywhere connected to the Internet via either databases, data collectors or integrators, display services, and even other devices.

Figure 1-1. *The future of IOT—all devices, everywhere[2]*

---

[2]https://pixabay.com/en/network-iot-internet-of-things-782707/.

IOT Is More Than Just Connected to the Internet

So if a device is connected to the Internet, does that make it an IOT solution? That depends on whom you ask. Some believe the answer is yes. However, others (such as myself) contend that the answer is not unless there is some benefit from doing so.

For example, if you connected your toaster to the Internet, what could be the benefit of doing so? It would be pointless (or at least extremely eccentric) to get a text on your phone from your toaster stating your toast is ready. So in this case, the answer is no. However, if you have people such as irresponsible teenagers or perhaps older adults whom you would like to monitor, it may be helpful to be able to check to see how often they use their toaster and when. That is, you can use the data to help you make decisions about their care and safety.
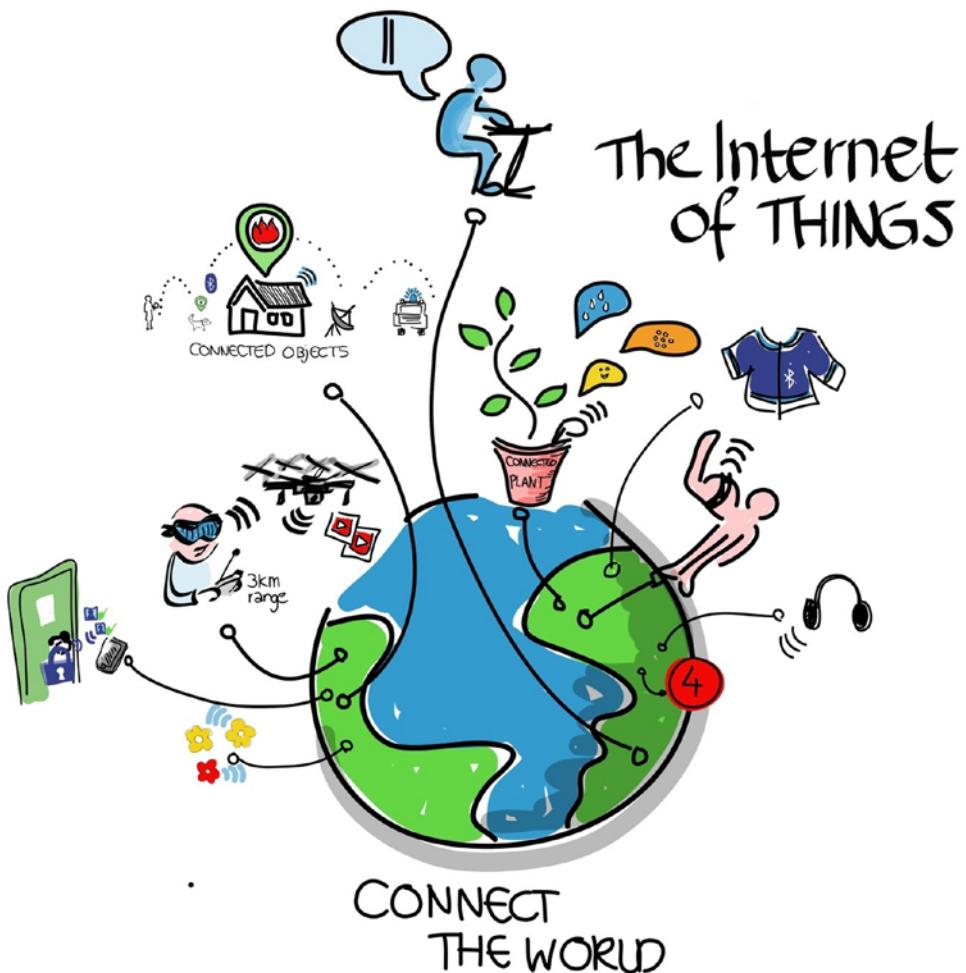
Allow me to illustrate with another example. I was fortunate to participate in a design workshop held on the Microsoft campus in the late 1990s. During our tour of the campus, we were introduced to the world's first Internet-enabled refrigerator (also called a *smart refrigerator*).[3] There were sensors in the shelves to detect the weight of food. It was suggested that, with a little ingenuity, someone could use the sensors to notify their grocer when their milk supply ran low, which would enable people to have their grocery shopping done not only online but also automatically.[4] This would have been great if you lived in a location where your grocer delivers but not very helpful for those of us who live in rural areas. While it wasn't touted as an IOT device (the term was coined later), many felt the device illustrated what could be possible if devices were connected to the Internet.

Thus, being connected to the Internet isn't what IOT is about nor is it a new concept. Rather, IOT solutions must be those things that provide some meaning—however small that benefit is to someone or some other device or service. Figure 1-2 depicts this a bit more clearly than Figure 1-1.

---

[3]https://en.wikipedia.org/wiki/Internet_refrigerator.
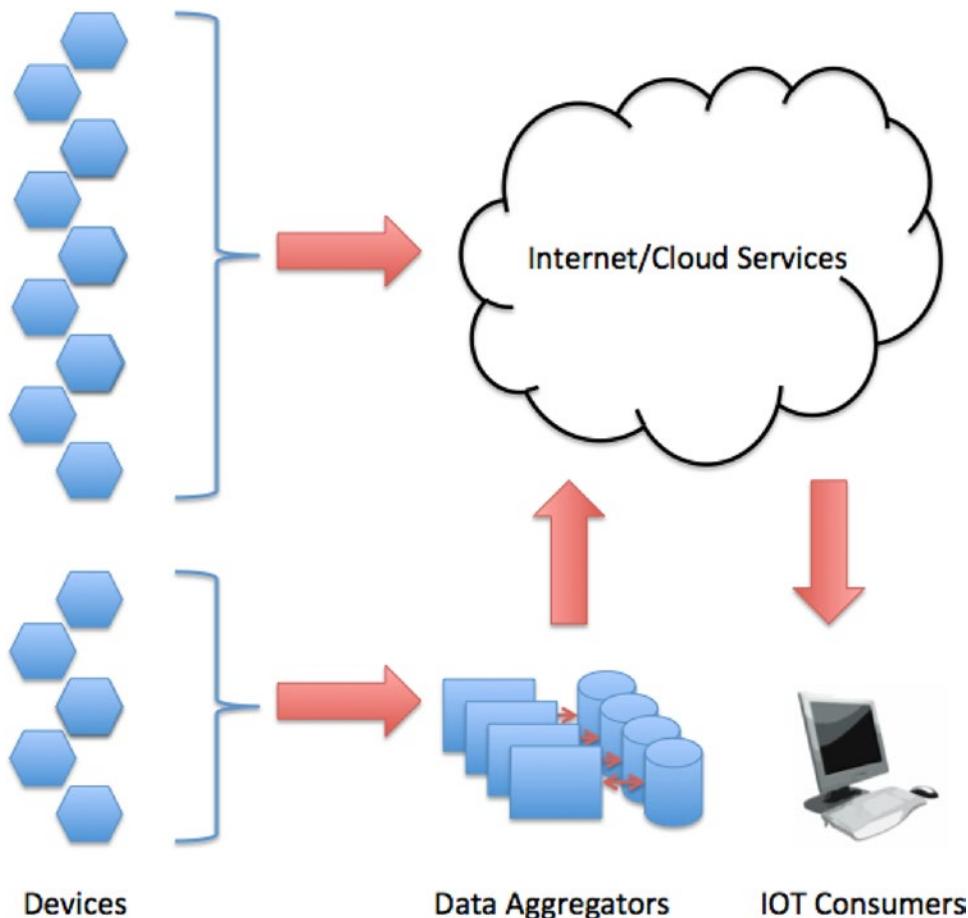
[4]Many businesses have automated reordering features built into their software. Most are triggered by a software or database event (such as low quantity), while sensors in the storage units trigger others.

Figure 1-2. *"Connect the world" by Wilgengebroed on Flickr,[5] via Wikimedia Commons*

Notice here you see things connected to the Internet in logical groupings. Observe the connected plants. The drawing depicts several sensors, but each sensor isn't necessarily connected directly to the Internet. It is more likely and more practical to connect the sensors from one or more plant to an intermediate node that sends the data either to a service on the Internet or perhaps to another node in the network for later processing. Figure 1-3 shows how this would look in a logical form.

Figure 1-3. *How IOT devices connect to the Internet*

On the left side of Figure 1-3 are the IOT devices. These could be a simple sensor, an entire sensor network, a device with one or more sensors, an embedded microcontroller solution with sensors, a more sophisticated microprocessor-based solution, or even a device such as a microwave oven, alarm clock, or television. Some devices will connect directly to the Internet, as shown at the top of Figure 1-3. These devices are the more sophisticated devices with built-in networking capabilities. At the bottom are devices that will connect to intermediate nodes such as data aggregators, computers, and so on, that filter, augment, and store the data prior to either presenting it via an Internet connection or sending the data to cloud services. At the far right is a user accessing the data via the Internet or a cloud service. Naturally, this can be any type of device such as a laptop, desktop, tablet, phone, watch, or other smart device or appliance (including another IOT device).

IOT Services

Sadly, there are companies that tout having IOT products and services that are nothing more than marketing hype—much like what some companies have done by prepending "cloud" or appending "for the cloud" to the name. Fortunately, there are some really good products and services being built especially for IOT. These range from data storage and hosting to specialized hardware.

Indeed, businesses are adding IOT services to their product offerings faster than anyone can keep up. And it isn't the usual suspects such as the Internet giants. I have seen IOT solutions and services being offered by Cisco, AT&T, HP, and countless startups and smaller businesses. I use the term IOT *vendor* to describe those businesses that provide services for IOT solutions.

You may be wondering what these services and products are and why someone would consider using them. That is, what is an IOT service, and why would you decide to buy it? The biggest reason you may decide to buy a service concerns cost and time to market.

If your developers do not have the resources or expertise and obtaining them would require more than the cost of the service, it may be more economical to purchase the service. However, you should also consider any retooling necessary in the decision. I once encountered a well-meaning and well-documented contracted service that permitted a product to go to market sooner than projected at a massive savings. Sadly, while the champions of that contract won awards for technical achievement, they failed to consider that the systems had to be retooled to use the new service. More specifically, it took longer to adopt the new service than it would have to write one from scratch. So instead of saving money, the organization spent nearly triple and was late to market. Clearly, you must consider all factors.

Similarly, if your time is short or you have hard deadlines to make your solution production ready, it may be quicker to purchase an IOT service rather than create or adapt your own. This may require spending a bit more, but in this case, the motivation is time and not (necessarily) cost. Of course, in reality it is a mixture of both cost and time.

So, what are some of the IOT services available? The following are a few that have emerged in the past few years. It is likely more will be offered as IOT solutions and services mature.

- *Enterprise IOT data hosting and presentation*: Services that allow your users to develop enterprise IOT solutions such as connecting to, managing, and customizing data presentation in a friendly form such as graphs, charts, and so on. Example: Xively (https://xively.com/).

- *IOT data storage*: Services that permit you to store your IOT data and get simple reports. Example: Sparkfun's IOT Data service (https://data.sparkfun.com/).

- *Networking*: Services that provide networking and similar communication protocols or platforms for IOT. Most specialize in machine-to-machine (M2M) services. Example: AT&T's cellular global SIM service (http://business.att.com/enterprise/Family/mobility-services/internet-of-things).

- *IOT hardware platforms*: Vendors that permit you to rapidly develop and prototype IOT devices using a hardware platform and a host of supported modules and tools for building devices ranging from a simple component to a complete device. Example: Intel's IOT gateway development kits (http://intel.com/content/www/us/en/embedded/solutions/iot-gateway/development-kits.html).

HOW TO RAISE CAPITAL FOR DEVELOPMENT: KICKSTARTER

When developing new solutions, it can sometimes be a case of needing money to make money. However droll that sounds, most developers are not independently wealthy nor do they have the funds to sink into the tooling and production of mass production for their devices. Fortunately, the Internet provides a mechanism for developers to raise capital needed to take their ideas to market.

While there are several sites that offer similar services, Kickstarter (http://kickstarter.com) has provided a revolutionary way to raise funds. You can post your idea on the site and offer your product or services for a small donation. Indeed, most successful Kickstarter campaigns offer the contributor rewards commiserate with the donation. For example, for a small donation you may get a T-shirt or the chance to buy the product at a discount. For a larger donation, you may get one of the first production units, free upgrades to newer models, and even a stake in the profits.

Not only does Kickstarter allow developers to raise capital, it allows individuals to participate in funding a project at a much more modest monetary commitment than a typical venture capitalist. If you're interested, see the Kickstarter home page and browse the hundreds of available campaigns. You never know, you may find something you want to invest in.

To give you an idea of what is possible, take a look at the Kickstarter campaign for the Kossel Pro (http://kickstarter.com/projects/ttstam/openbeam-kossel-pro-a-new-type-of-3d-printer). The developers more than doubled their monetary goal, their product has become a reality, and orders are coming in fast and furious. I should know; I own one of the first production units!

Example IOT Solutions
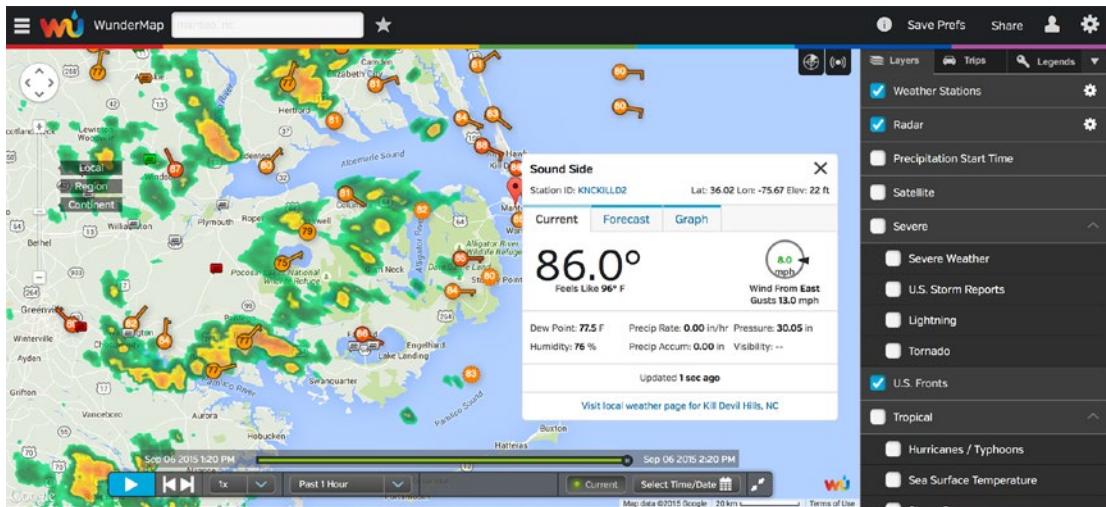
Let's take a look at some example IOT solutions. The IOT solutions described in this section are a mix of solutions that should give you an idea of the ranges of sizes and complexities of IOT solutions. I also point out how some of these solutions leverage services from IOT vendors.

Sensor Networks

Sensor networks are one of the most common forms of IOT solution. Simply stated, sensor networks allow you to observe the world around you and make sense of it. Sensor networks could take the form of a pond-monitoring system that alerts you to water level, water purity (contamination), and water temperature; detects predators; or even turns on features automatically such as lighting or feeding the fish. If you or someone you know has spent any time in a medical facility, chances are a sensor network was employed to monitor body functions such as temperature, cardiac and respiratory, and even movement. Modern automobiles also contain sensor networks dedicated to monitoring the engine, climate, and even in some cars road conditions. For example, the lane-warning feature uses sensors (typically a camera and microprocessor and software) to detect when you drift too far toward lane or road demarcations.

Thus, sensor networks employ one or more sensors that take measurements (observations) about an event or state and communicate that data to another component or node in the network, which is then presented in some form or another for analysis. Let's take a look at an example of an intriguing alternative weather service.

Weather Underground (http://wunderground.com) is a community-driven site that permits amateur and professional weather enthusiasts around the world to connect their weather station to the Internet and share the data that their sensors provide. This means you can get the latest weather information for your region, city, and even locality simply by clicking the map and zooming in. You will see icons representing each local weather station (which displays local temperature) that you can click to see more information. Figure 1-4 shows Weather Underground's WunderMap that uses Google Maps to display the weather stations. As you can see, you can click any of the weather stations to see more information.

Figure 1-4. *Weather Station Network (courtesy of* http://wunderground.com/wundermap*)*
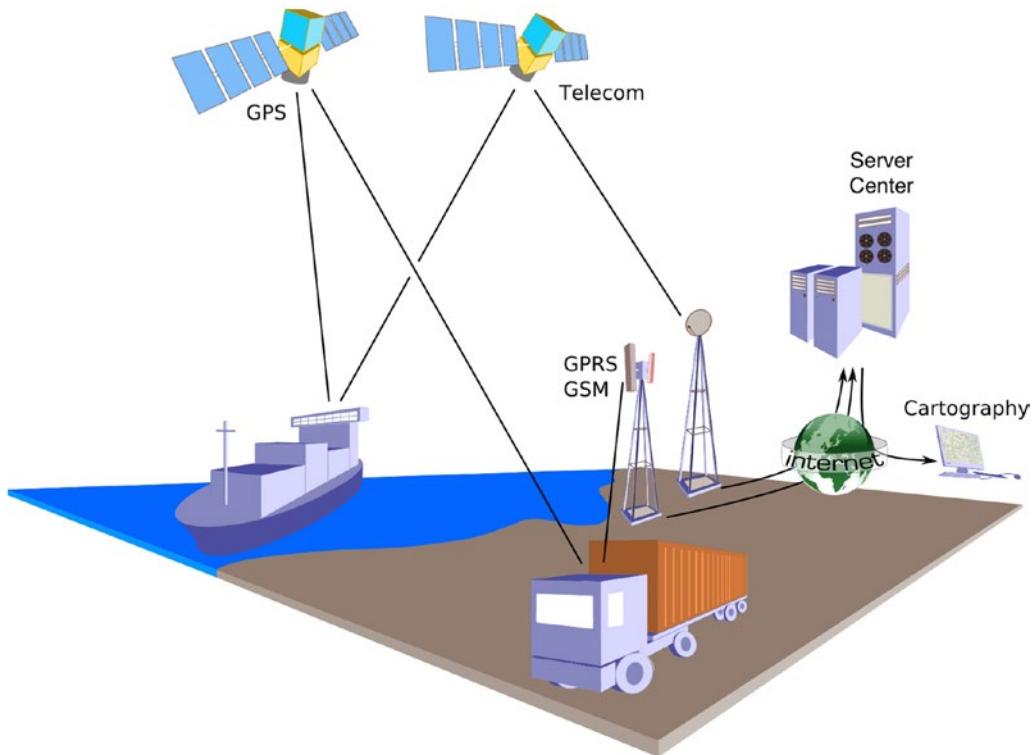
You can do far more than this with the WunderMap. As you can see, the map also shows radar data that you can use to see where precipitation may be occurring. You can also see the map in motion replaying data from previous updates from the radar and weather stations. Weather Underground is a really great site for those interested in weather—whether you have your own weather station or not, you can get a ton of information from the site.

This is perhaps one of the best examples I've found to illustrate the power of IOT and sensor networks in particular. Not only are you able to see the data generated by the sensors in your neighbor's weather station, you can also see the data from dozens more stations from around the area, the state, or even the country! This is the true power of IOT materialized.

Fleet Management

Another example of an IOT solution is a fleet management system (https://en.wikipedia.org/wiki/Fleet_management). While developed and deployed well before the coining of the phrase IOT, fleet management systems allow businesses to monitor their cars, trucks, ships—just about any mobile unit—not only to track their current whereabouts but also to use the location data (GPS coordinates taken over time) to plan more efficient routes, thereby reducing the cost of shipment.

Fleet management systems aren't just for routing. Indeed, fleet management systems allow businesses to monitor each unit to conduct diagnostics. For example, it is possible to know how much fuel is in each truck, when its last maintenance was performed (or more importantly when the next maintenance is due), and much more. The combination of vehicle geographic tracking and diagnostics is called *telematics*. Figure 1-5 illustrates a fleet management system.

Figure 1-5. *Fleet management example[6]*

In this figure, you will see how GPS systems can track location as well as satellite communication to transmit additional data such as diagnostics, payload states, and more. All of these ultimately traverse the Internet, and the data becomes accessible by the business analysts.

You may think fleet management systems are only for large shipping companies, but with the proliferation of GPS modules and even the microcontroller market, anyone can create a fleet management system. That is, they don't cost millions of dollars to develop. For example, if you owned a bicycle delivery company, you could easily incorporate GPS modules with either cellular or wireless connectivity on each delivery person to track their location, average travel time, and more. More specifically, you can build a GPS tracking solution with an Arduino and a small supporting set of electronics.[7] In fact, I propose such a solution could be used to minimize delivery times by allowing packages to be handed off from one delivery person to another rather than having them return to the depot each time they complete a set of deliveries.

---

[6]CC BY 2.0 (http://creativecommons.org/licenses/by/2.0).
[7]See http://makezine.com/projects/make-37/gps-cat-tracker-2/ for information about how to build a small GPS tracker for pets. The technology would be the same or similar.

9

Home Automation

Another IOT solution that is becoming more prevalent (or at least more popular) is home automation[8] (also known as a *smart home*). While not new in any sense (home automation has been around for quite a while), home automation solutions have become more interesting now that many vendors are making them Internet ready. Indeed, most solutions provide either direct access from the Internet or access through cloud services.

A home automation solution is typically a set of sensors, actuators, and devices that allow you control things in your home. You can find sensors to detect movement, the state of windows and doors (open/close), temperature, humidity, and more. You can also find actuators such as locks that allow you to unlock or lock your doors remotely, open and close garage doors, and even turn lights on and off. Finally, you can find more sophisticated devices such as smart thermostats that can be programmed remotely, cameras that you can view and record images, robotic vacuum cleaners, and even phones that permit you to dial as if you were at home. While none of these is new, what is new is the packaging of these devices into an IOT solution.

For example, many home improvement stores such as Lowes and Home Depot carry their own line of home automation solution. You can find ready-to-install devices such as door locks, cameras, thermostats, and more that you can quickly and easily install and, with the aid of the included software, access remotely. There is even a device that allows you to turn your kitchen faucet on and off!

The Lowes solution, called Iris (`http://irisbylowes.com`), is a subscription service combined with a special network hub (called the Iris smart hub) to which all the devices connect. To access the data generated and features such as home security, pet monitoring, and more, you use an application (*app*) for either an Android or IOS device. The subscription has several levels ranging from a free basic service that allows you to connect to devices for basic services such as locking and unlocking doors, detecting movement, viewing short videos from cameras, and more. The paid service allows you much more access to devices and to control them remotely such as scheduling device power (turning lights on while you are away randomly to make it appear you are home) and more. Iris is sold as a starter kit with a few devices that you can expand as your budget or needs permit. See the Iris page for more information.

The Home Depot solution (`http://homedepot.com/c/Home_Automation_Basics`) also uses a hub (called Wink), but unlike the Lowe's solution, the devices you can add do not need to be from a single vendor. In fact, you can add devices that communicate via Wi-Fi, Bluetooth LE, Z-Wave, ZigBee, Lutron, ClearConnect, and Kidde. Thus, you can mix and match your devices with more freedom to grow your solution to your needs. Like the Lowe's solution, you can monitor all manner of things and view the data via an app called the Wink app. I should also add that some of the home automation devices available from Home Depot can operate without the hub, but the hub is required for remote access. Like Lowe's, you can purchase starter kits that are easy to set up and use. See the Home Depot Home Automation page for more details and links to the kits available.

The Lowe's and Home Depot solutions are just two of many available. A quick Google search will result in dozens more that you can choose—some that are proprietary like the Lowe's solution and others that are more open or can be expanded by devices from multiple vendors.
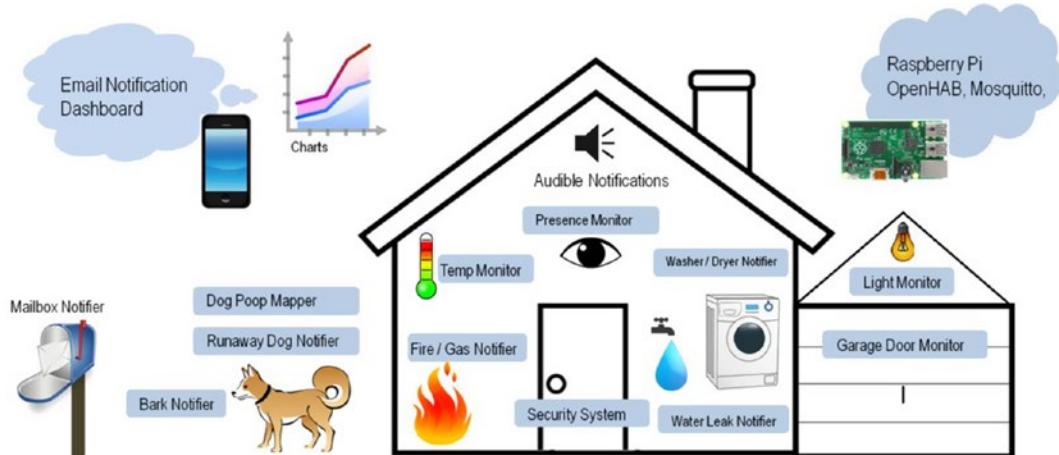
You can also build your own home automation solution using microcontrollers such as Arduino, Raspberry Pi, and BeagleBone. There are a number of books on the subject ranging from simple solutions to complex solutions. The following are just a few of the growing number of DIY home automation books.

- Steven Goodwin, *Smart Home Automation with Linux and Raspberry Pi* (Apress, 2013)

- Marco Schwartz, *Arduino Home Automation Projects* (Packt Publishing, 2014)

- Onur Dundar, *Home Automation with Intel Galileo* (Packt Publishing, 2015)

---

[8]`https://en.wikipedia.org/wiki/Home_automation`.

You can also find instructions for creating home automation solutions on the Internet. One example of a complex solution is the article (called an *instructable*) by Eric Tsai at `http://instructables.com/id/Uber-Home-Automation-w-Arduino-Pi/`. In his article, Eric describes a foundation for a home automation solution that you can build yourself from easy-to-obtain and easy-to-use components. Figure 1-6 shows an example of Eric's article that depicts what is possible with a little imagination.

Figure 1-6. *DIY home automation (courtesy of Eric Tsai,* `http://etsai.net`*)*

As you can see, with a DIY home automation solution, you can build it however you like. You can add Arduino-controlled sensors that communicate via ZigBee modules, Raspberry Pi boards to monitor cameras, custom electronics to control garage doors, and more! In fact, if you aspire to be a maker or are an accomplished maker, chances are you can modify your existing devices to add remote capabilities. For example, garage door openers are easy to add a wireless module to in order to send signals over the local WiFi (or even the Internet). Even if you aren't an experienced software developer, you can use services such as Xively (`http://xively.com/`) to send your home automation data for monitoring. Of course, if you invest a little time in developing a simple web page, you can connect your home automation solution through your local network. While it may not be as fancy as the commercial units, you can make it do whatever you want.
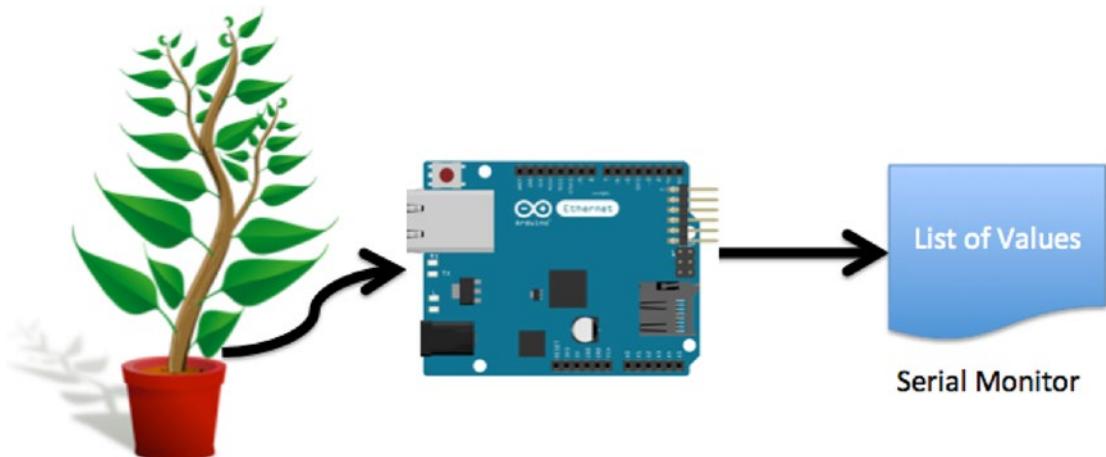
Now that you know what IOT solutions are and have seen some examples, let's discuss what is arguably one of the most critical components of an IOT solution: data.

What Is IOT Data?

Whether your IOT solution is keeping watch on your breakfast burrito while you finish your shower or you're relying on instruments to pilot your boat to safety during a storm, the data produced and acted on is the most important artifact and indeed the lifeblood of the IOT solution.

Why? Because the solution is meaningless without the data. For example, if you had an IOT solution that monitored your body functions but never stored the data, the most you could achieve is an instantaneous reading. Without storing the data, you cannot perform any diagnostics from events in the past. Clearly, the data is important.

Perhaps equally important is how that data is stored and protected from exploitation or misuse. This is an especially important aspect of IOT solutions, which I will discuss in more detail later. For now, let's consider data for IOT solutions starting with a simple plant-monitoring system. Figure 1-7 shows a simple solution with a single plant. While a good solution would also include a light sensor to monitor how much light the plant is receiving, this one has been simplified so you can see the data produced.

Figure 1-7. *Plant monitoring with Arduino*

In this example, you see an Arduino with a soil moisture sensor connected. The Arduino generates a list of readings from the soil moisture sensor that is placed in the soil of the plant. The code to do this is not complicated, as shown in Listing 1-1.

Listing 1-1. Plant Soil Monitoring

```
/*
 Simple Plant Monitor Example

 Display the value of the soil moisture sensor and threshold status.

*/

// Thresholds for sensor to detect wet/dry conditions. Adjust these
// to match your soil characteristics.
int upper = 400;
int lower = 250;

// Pin for reading the sensor
int sensorPin = A0;

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while (!Serial);
```

```
  Serial.println("Simple plant monitor");
  Serial.println("raw value, moisture state");
}

void loop() {
  int value;

  // Read the sensor
  value = analogRead(sensorPin);

  Serial.print(value);
  Serial.print(",");

  // If value is less than lower threshold, soil is dry else if it
  // is greater than upper threshold, it is wet, else all is well.
  if (value <= lower) {
    Serial.print("dry");
  } else if (value >= upper) {
    Serial.print("wet");
  } else {
    Serial.print("ok");
  }
  Serial.println();
  delay(1000);
}
```

I kept this simple using the serial monitor as the output so that you can see the data generated. Listing 1-2 shows the output of a sample run where I watered the plant during the run. Notice in the listing of the output that the sensor correctly measured a change when I watered the plant (not a dramatic change in values). If you were building a more sophisticated (more useful) plant monitor, you would likely use some form of output such as an LCD panel or web server or you would store the data in a database.

Listing 1-2. Output of Soil-Monitoring Example

```
Simple plant monitor
raw value, moisture state
159,dry
217,dry
225,dry
224,dry
225,dry
225,dry
226,dry
248,dry
249,dry
256,ok
261,ok
279,ok
276,ok
254,ok
266,ok
295,ok
```

13

```
291,ok
302,ok
394,ok
467,wet
506,wet
419,wet
```

■ **Note**   You can download the source code examples for this book from the Apress web site.

For purposes of this illustration, the text output is sufficient. This is because I want to call your attention to the rows of data. The first column is the raw value as read from the sensor. Clearly, this data has little human-readable information. After all, it's just a numeric value for the analog signal. That is why I used threshold values to determine or qualify the data. You can see this as the values change and the moisture level rises.

As you can see, the data that makes most sense to us is the "dry," "wet," and "ok" values. The raw values are not really that interesting. This is an excellent albeit simplistic illustration of how the raw data from a sensor needs additional augmentation to make it useful. However, I must also point out that if you stored only the derived values, if you need to adjust your thresholds, you cannot reevaluate the derived values. For example, if you determine the upper threshold needs to change and you want to do some analysis on how the change would have affected readings in the past, since you have only the "dry," "wet," and "ok" values, you cannot perform this analysis. Thus, saving the raw data is always a good practice.

■ **Tip**   Always save the raw data as well as the calculated or derived values. You never know when you will need it.
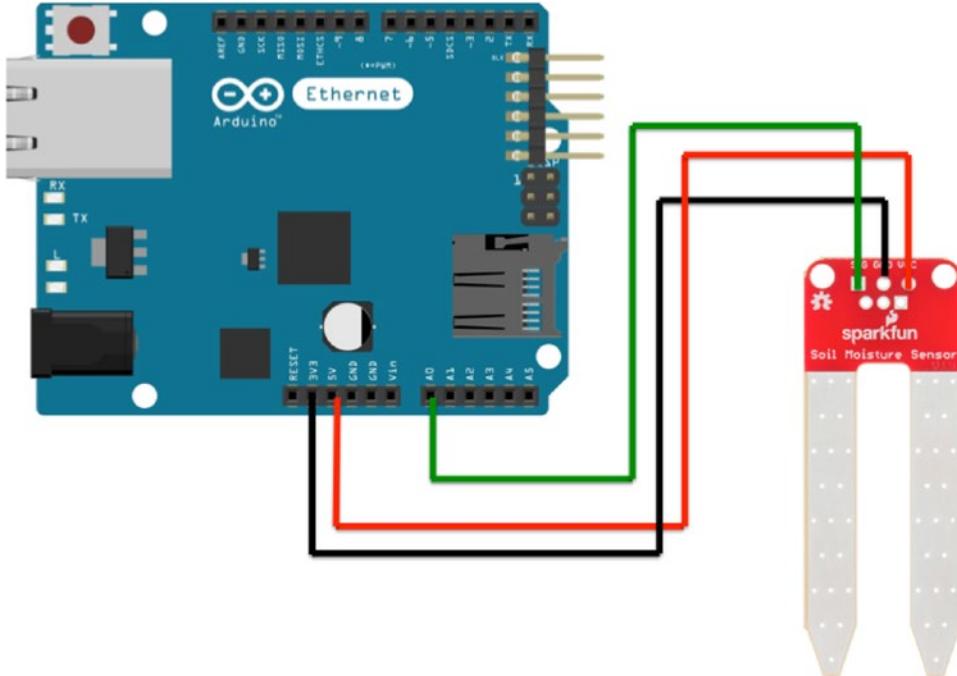
If you are interested in building this example or perhaps embellishing it with an LCD or LED to warn when the plant needs water or better still to automatically water the plant with a servo or stepper motor and a water source, I encourage you to do so. It is a fun project. You can even modify the code to support a web server that you can use to remotely check the status of your plants.[9] The parts are easy to find and available from most online electronics stores such as SparkFun (http://sparkfun.com), Maker Shed (http://makershed.com), and AdaFruit (http://adafruit.com). I've included a couple of links to articles ordered from simple to complex that explain how to build this project and similar projects. Figure 1-8 shows how the sensor is wired to the Arduino.

- Sparkfun's soil moisture tutorial (https://learn.sparkfun.com/tutorials/soil-moisture-sensor-hookup-guide?_ga=1.98811421.2053037341.1391972341)

- Soil moisture with Grove components (http://seeedstudio.com/wiki/Grove_-_Moisture_Sensor)

- Make's potted plant protector (http://makezine.com/projects/potted-plant-protector/)

---

[9]Hint: See the web server example under the Ethernet category in the Arduino IDE.

- Self-watering plant (http://instructables.com/id/Self-Watering-Plant/)

- Adafruit's wireless garden tutorial (https://learn.adafruit.com/wireless-gardening-arduino-cc3000-wifi-modules)

Figure 1-8. *Wiring the simple plant monitor*

There is another reason I wanted to show you this data. IOT solutions often employ intermediate nodes in the network. More specifically, it is often the case that a sensor is installed or connected to a much smaller set of electronics such as a microcontroller or even a simple integrated circuit. This node would then take the sensor value(s) and send them to another node elsewhere on the network. It is important to note that this level can use a networking protocol other than Ethernet or WiFi to simplify and reduce the need to have a unique address for each device. These devices typically do not have enough resources to support the more complicated networking protocols and thus require something more lightweight that can be achieved with limited resources.

When data is exchanged between nodes like this, it is called *machine-to-machine* data exchange and often transmitted in the raw form. There are several reasons for this. Most notable is to save memory and help speed communication, especially for small microcontrollers and similar embedded processors. That is, it is far faster to send a single integer or even a floating-point value than a formatted text string. This could be critical if the nodes at the sensor level use a communication protocol that operates with lower resources (memory) such as XBee modules.

With all this talk of the data in IOT solutions, one must wonder where it all goes and how this data will accumulate.

15

IOT Predictions: Data Overload?

You don't have to look far or long on the Internet to discover some dire predictions about the IOT.[10] Most of this stems from the proliferation of new devices and new ways to connect devices to the Internet. This presents two problems: the need for more address space (IP addresses) and addressability and the need for better ways to manage extremely large amounts of data (big data). Less popular but equally important is the security of IOT data, devices, and services. I address each of these in the following sections.

Addressing IOT Devices

Just because you can tack on an address to a random spot on the planet, it isn't necessarily easy to find. That is, if you mark a rock somewhere in the desert with a bright red "X" or simply write a number on it, no one is going to find it. Even if people spend the time to comb the desert, they will need a hint as to its general area to find it. Of course, you could add metadata such as latitude and longitude or even a GPS homing beacon, but the "X" you placed on the rock is clearly not enough information. This illustrates two issues for addressing IOT devices: having a way to uniquely identify the device and finding or addressing the device.

Are There Enough Addresses Available?

There is a legitimate concern that shortly the number of IOT devices will quickly exceed the maximum capacity of IP addresses available. Currently, the IPv6 protocol[11] allows for approximately 3.4×10^{38} addresses. That is 340 undecillion numbers! While that is a massive number, it is likely the usable range of public addresses will be considerably less but still in the undecillion level. This is great because some predict the number of IOT devices in the future to number in the billions or perhaps even hundreds, thousands, or millions of billions.

Even if the public addressable IPv6 addresses number half or even a fourth of the available IPv6 addresses, we won't run out any time soon. Indeed, it is possible (and some make claims that assure us) that IPv6 provides ample addressability for all conceivable IOT devices in the future, but having all of those devices addressed does not mean they will be easily found.

How Can You Find an IOT Device?

Assigning an IP address to an IOT device doesn't necessarily make it easy to find. Indeed, if every IOT device had its own IP address, all would be connected to the Internet and potentially each other, but how would you find the device you're looking for? If they are your devices or someone you know who is willing to share their address, you can find them by knowing the right information. But what if you wanted to know whether anyone else in your neighborhood had an outdoor camera? Suppose you needed to access their imagery to help solve a crime or identify the wanderings of a stray animal? Short of knocking on your neighbors' doors, there is no easy way to find these IOT cameras.

---

[10]I saw a bumper sticker once that read, "The Internet is full. Go play outside." While whimsical, the slogan contains a small grain of truth and a dollop of advice for the younger generations.
[11]https://en.wikipedia.org/wiki/IPv6.

A SIMPLE SEARCH IS NOT ENOUGH

It is not enough to merely search the available IPv6 addresses looking for an IOT device. Some have estimated, even with a modestly fast search engine, that it could take many years and perhaps even thousands of years to search and identify all IPv6 devices connected to the Internet. And that doesn't include the ones being added or removed daily. Clearly, we cannot simply add IPv6 capability to every electronic device or thing that moves (and some that don't) in our lives and expect to be able to access them without more knowledge of what they are, where they are, and what they do.

Like with the rock in the desert, you need more information. You need to know how that device is connected, what it does, how it is used, and what data it produces. Thus, you need some form of broker or service that tracks the device. Perhaps a more poignant question is, why do you need to know or access the device in the first place? Isn't it more likely that you would treat your neighbors' homes as services that you can request data? For example, if you could use Google Maps to find your neighborhood and click the homes around you to see what IOT data is publicly available, would that not be much more useful than trying to find a specific IOT device (camera)? Doesn't this sound familiar? It should. This is exactly how the WunderMap works in the Weather Underground site.

In this case, each home would be an IOT service provider generating data. Some data could be made public such as externally facing cameras, while other data may be private and require secure logins in order to access the devices in the home. Recall the home automation use case. Consider how convenient it would be to be able to check on your home remotely or perhaps grant permission to a babysitter to watch TV or use a device in the kitchen.

No matter what the IOT service is, the fact remains that it is unlikely that you would need to access an IOT device directly. The service can provide all the functionality needed (or permitted). Not only does this dramatically reduce the search problem, it also helps limit the number of publically addressable IOT devices. That is, devices "behind" or "inside" the IOT service do not need to be made public. Even more importantly, this allows you to firewall or secure your IOT devices in a more robust manner. Think of what it would mean to discover that your IOT camera could be accessed by anyone anywhere with a simple hack.

By placing the IOT devices behind an IOT service (or broker, application, and so on), you also permit the use of lower-resource communication protocols. That is, the IOT devices can be built from cheaper and more limited hardware. For example, you do not need a laptop to monitor a plant sensor. Furthermore, if you had a home automation solution that permitted you to connect a plant monitor that uses XBee protocols, you could build that plant sensor using much less hardware and therefore more cheaply.
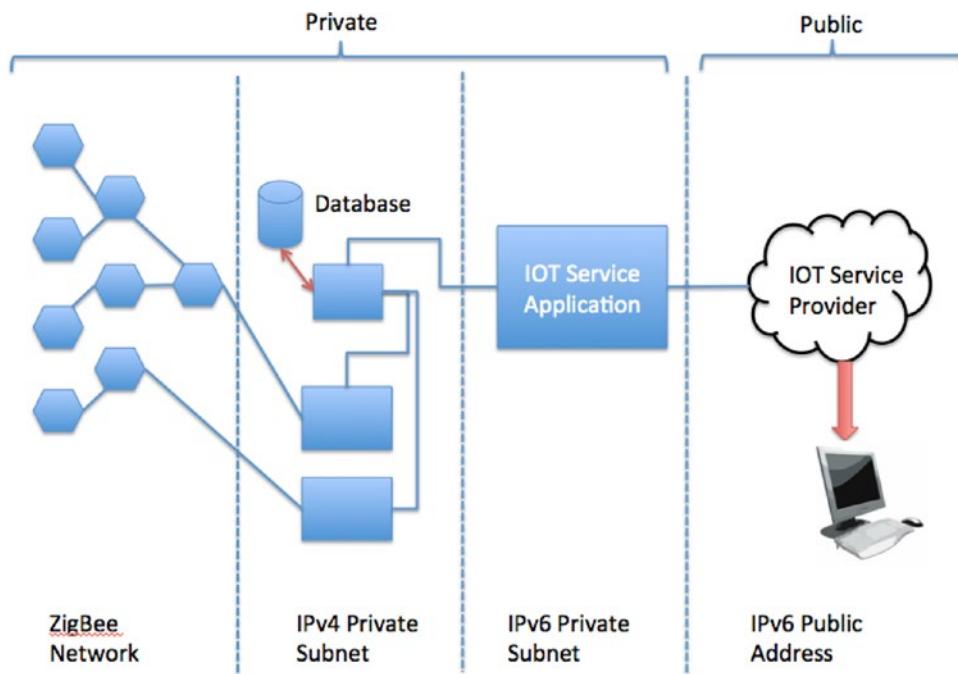
Smaller or lower-resource hardware and communication protocols solve another issue with IOT devices and sensors in particular. Sensors are not generating data at a preset interval. While some sensors contain timer circuits or can generate a value only periodically, sensors are generally polled at certain intervals by another device (such as the Arduino, Raspberry Pi, and so on).

Furthermore, communication protocols such as XBee are not lossless mechanisms. Indeed, however seldom, data loss is likely and the protocols support complete loss of a packet. If your sensor is generating values 30 times a minute, does it really matter if you receive only 29 values instead of 30? Perhaps some solutions may require greater precision, but for sensors that monitor events, this is acceptable, and it fits the model for sensors quite well. It allows for the possibility that the value is not ready; the sensor has not changed its state, and so on, permitting a much broader range of sensor behaviors that you can support.

The solution I am describing here has been labeled in a number of ways, but the most correct and indeed the most profound proposed architecture for the IOT is called Chirp. Francis daCosta describes this architecture in his book *Rethinking the Internet of Thing*s (Apress, 2013).

Chirp is simply the name of a small packet of data that contains only the minimal information needed to convey the data to its destination. This fits the small overhead requirement for lighter protocols and also permits for occasional loss of a packet. At the lowest point in the IOT solution, devices use protocols that support chirps to be sent to nodes that can monitor the data and take appropriate action. For example, the node may be built to take 30 samples generated in a minute and average them over time (say every five minutes) to generate a smoother data sample. I've seen this technique employed for solutions that used less accurate sensors where spikes (high or low) beyond a threshold were not used, thus allowing for a smoother reading from the sensor at a slower rate. It is less data but potentially more accurate.

Thus, the IOT solution would be built with several layers. At the lowest layers, IOT devices are networked to nodes that collect or process the data, which are then connected to devices that provide or enable certain services. For example, an IOT camera could be connected to an intermediate node that takes commands from a node at a higher layer. Similarly, a sensor network could be employed and monitored using several data aggregation and data-processing nodes that send the data to a database, which is then accessed by nodes in the higher layers. Figure 1-9 shows an example of this concept.

Figure 1-9. *Concept of layered IOT solution*

Notice at the top I've marked the layers behind the IOT service as private. This is to indicate that these layers and the devices within them are not directly accessible from the Internet. That doesn't cease to make them IOT devices; on the contrary, it helps define an architecture for layering the solution to permit optimization by using only the necessary protocols (and hardware needed for each layer). For instance, the sophistication of the devices, security practices, and programming increases from left to right in the drawing.

For example, on the left side of Figure 1-9 is a set of sensors or lightweight IOT devices connected using a ZigBee network. This could take the form of small devices employing XBee modules for sending data to the next layer, which contains the data aggregation and database nodes. At this layer, you could employ a private IPv4 Ethernet network. Why IPv4? Because most small devices like Arduino and Raspberry Pi support it

natively and by default. Thus, you can begin to connect these nodes with an application node that provides a gateway to the cloud-based services. At this point, you could continue to use IPv4 or switch to IPv6 since this is likely to be a more sophisticated server. On the right side of Figure 1-9 is a depiction of the IOT service (or services) you would then employ to grant access to your IOT solution. This could be anything from a front-facing web server to a complete IOT solution provided by commercial IOT vendors.

Clearly, an architecture like this permits the lower layers (those on the left) to communicate with lower-resource protocols and even deal with the occasional loss of data, while the higher layers (those on the right) communicate with more complex protocols. This also allows you to place the more complex parts of the solution on the layers or devices that are most appropriate for the task. For example, while you can connect your Arduino to a MySQL database server,[12] it is unlikely you would host it on the same platform (but you could on a Raspberry Pi) as other services. Finally, it permits you to design the appropriate level of security into each layer.

Now that you understand the addressing issue with IOT devices, let's talk for a moment about the data. With billions of IOT devices generating data, where is it all going, and how will you access it?

IOT and Big Data

Another concern IOT experts have is how quickly and how much the size of the data generated by IOT solutions will grow. That is, as more and more IOT devices are added and the data is archived, the size of that data will grow exponentially. As the data becomes sufficiently large, it ceases to be feasible to access it using traditional database access mechanisms. For example, accessing the sensor data for all the thermostats located in the United States could eventually become an absurdly large number (rows, bytes, and so on). Even if you had a reason to see this data, the amount of data would be overwhelming. If you narrowed it down by state, the data could still be more than is possible to search or for that matter retrieve.

WHAT IS BIG DATA?

Big data[13] refers to the relative size of data that will be processed, analyzed, viewed, or otherwise manipulated to draw conclusions (for example, data analysis, data warehousing, and so on). The relative size refers to the characteristic that big data exceeds the capacity of most computing platforms to contain or otherwise process the data in a reasonable amount of time. That is, it is more than a single system or even a complex system can handle.[14]

There are many approaches to handling big data, but most solutions use dozens to thousands of computing platforms to divide and conquer the problem. Two notable examples are Oracle's Big Data offerings (`http://oracle.com/big-data/index.html`) and MySQL + Hadoop (`http://mysql.com/why-mysql/white-papers/mysql-and-hadoop-guide-to-big-data-integration/`). While these solutions are differentiated based on customer or use cases, they both solve the problem (at a high level) of leveraging distributed database and execution to process the data.

---

[12]`https://github.com/ChuckBell/MySQL_Connector_Arduino`.
[13]`https://en.wikipedia.org/wiki/Big_data`.
[14]For example, it should not take 7.5 million years to get an answer of 42.

Thus, some pundits propose the IOT will produce data that is large and can be used only with big data solutions. For example, if you wanted to analyze patterns of use for a certain class of IOT devices across a geographical region (state, country), even though each IOT device may have its data hosted separately, there is a need to aggregate the data for compilation and processing. Thus, even if each device had only a small amount of data, aggregating the data over millions or even billions of IOT devices could generate a big data crisis.

Furthermore, while I and others propose a layered approach to IOT solutions and most will likely be built to host its own data, it is still probable at some point we will want to search and mine data across similar IOT solutions. While the thermostat example is a bit fictitious, it may be more likely that you need to research data from these solutions to develop patterns such as temperature fluctuations, fuel usage during peak weather months, or even evaporation rates of ponds and reservoirs to help predict water consumption and conservation rates. In this case, it is likely you would have to mine data from several repositories.

Thus, even if the data were compartmentalized so that there isn't a single database that holds everything,[15] the data you've mined will likely require temporary storage for analysis, making the data analysis a case of working with large (big) data. However it comes to pass, it is true that at some point acquiring, aggregating, and processing data from the IOT will require specialized big data solutions.

NOT EVERYONE AGREES

You may find it interesting that there are several schools of thought concerning big data. Some criticize its existence, others criticize the mechanisms we use to work with big data, and others insist the real solution is yet to be realized. Whatever the case, be advised that the landscape of big data is still evolving.

Fortunately, there are many vendors working on this problem, and the proliferation of cloud service providers ensures we will not have to create big data solutions ourselves. However, we will still have to deal with storing and making accessible our IOT solutions data—which is what this book is all about.

■ **Tip** If you'd like to know more about big data and how it relates to IOT, see Stackowiak, Licht, Mantha, and Nagode's book, *Big Data and the Internet of Things* (Apress, 2015).

Now that you understand that IOT data is no small matter and indeed will likely become a massive archive ripe for harvesting even more knowledge from the world around us, let's discuss the number-one concern beyond how to store the data—how to protect it and the IOT solutions from exploitation.

IOT Security

IOT developers also need to consider securing their devices, data, and services. Indeed, all solutions that use the Internet must develop better security practices. The unique aspects of an IOT solution make it especially difficult to plan and implement stringent security practices because of the multiple points of vulnerability. More specifically, each component may have different types of vulnerability, from physical access to sensors and IOT devices to remote attacks against the IOT services.

---

[15]Now that is absurd.

The recent rash of massive data breaches proves that security simply wasn't good enough. We've seen everything from outright theft to exploitation of the data stolen from well-known businesses like Target (more than 40 million credit card numbers may have been compromised) and government agencies like the United States Office of Personnel Management (more than 20 million Social Security numbers compromised). Interestingly, the source of the breach was traced back to third-party contractors and services. Clearly, no one is safe. We need a revolutionary step rather than refining the tried-and-true mechanisms.

Sadly, there is a limit to how far we can go in securing our solutions. As any information technology (IT) professional will tell you, applying the best, stringent password policies and tight security practices can force users to jeopardize the very strategy designed to protect them and their data. For example, consider password policies that require passwords to be 16 or more characters with at least four capital letters, six numerals, three special characters, and no English dictionary words; to expire every 60 days; and to not have more than seven characters in common with previous passwords.[16] In this situation, some users will be forced to write down their passwords because they cannot remember a random mixture of letters, capitals, numerals, and special characters.

However, making passwords harder to guess or crack is only one strategy. Indeed, there are various philosophies about how to secure systems properly. While an in-depth discussion of all techniques is beyond the scope of this book, it is important to consider one of the more popular philosophies we can use for our IOT solutions. It is the philosophy of using multiple components to identify individuals.

For example, a user may need to know a key phrase (password), possess an authorized tangible talisman (an RFID badge), and have on file a biometric signature (fingerprint). To gain access to one of the systems, the user would have to know the password, present a valid RFID badge, and have their fingerprint read and verified.

This may sound a little like science fiction or even super-secret spymaster work, but it ensures the access will be granted only to someone who meets all three components. That is, it may be possible to guess, crack, or simply steal a user's password, and it may be possible to acquire or even spoof an RFID badge; it even may be possible (however far-fetched) to copy someone's fingerprint. It is far more difficult to acquire all three components without compromising the identity of the user. However, the downside is the user cannot gain access unless she has all three components. While it is unlikely the user would lose their fingerprint (but injuries and skin conditions can make the reader fail), it is possible the user could lose or misplace their badge or forget their password. Thus, once again, the security practice is diminishing the user's experience and making it more difficult for the user to access the system.

So what do we do? Do we implement good practices to ensure the systems are not easily compromised, or do we risk lower security for ease of use? The bottom line is you must choose the security solution that best meets the need to protect the data and services without forcing the user to endure onerous practices and without making their lives difficult.

You may be wondering what this has to do with your Arduino-based sensor platform. After all, there isn't much someone can do to exploit an Arduino, is there? That depends on the Arduino and how it is connected. For example, a newer Arduino that supports common lightweight operating systems or that is connected to your network can be exploited. I won't expand on that, but suffice to say it is possible. The risk of exploitation increases with the sophistication of the IOT device. For example, in general, a Raspberry Pi may have more risk than a bare-bones sensor and XBee module. This is because the Raspberry Pi is capable of running Linux and therefore supporting all manner of hacking tools and utilities.

---

[16]No, really, this does happen. It is a perfect example of how good security practices can go wrong however well intended. That is, if the policy makes the users' lives so difficult that they must violate best security practices to cope, the policy has gone too far.

■ **Caution**   Whichever security philosophy or strategy you employ for your user-accessible devices, you still must consider securing the rest of the nodes in the network.

But it isn't just software that can be exploited. For example, placing an IOT device in an enclosure outside your home that is connected via Ethernet is vulnerable to hackers who gain access to the Ethernet cable. Granted, someone would have to know the IOT device exists, but the risk of exploitation is real. To combat this, you can employ lighter-weight hardware and more simplistic communication protocols[17] that cannot be easily hacked.

But is security really a concern for well-designed IOT solutions? Let's look at a recent example. One of the biggest automotive brands in the United States (and the world), Jeep, has recently come under fire for vulnerability in its infotainment[18] solution. A group of highly skilled hackers was able to remotely access the system and hack into the other electronics in the car. The group was able to sound the horn, turn on the wipers, and even affect the handling and brakes. Worse, this all happened while Andy Greenberg, the author of the article "Hackers Remotely Kill a Jeep on the Highway: With Me in It,"[19] was at the wheel! No, this is not a myth. It actually happened, and Jeep has issued not one but two recalls for security patches to its systems. So what does this say about the future of IOT-enabled cars? You had better be certain security is not only built in but done very well. Clearly, Jeep has some more work to do.

WHY SECURITY?

You may be wondering why we are discussing security in a book dedicated to databases in IOT solutions. You may have heard "charity begins at home," which means we must teach our children the morals and ethics of taking care of others through generosity. For IOT solutions, there is analogy that applies to security. We must build security into our IOT solutions from the beginning. That is, we must design with the overarching goal of protecting the data and access to it from exploitation or theft. Every component must have security design goals, from simple sensors connected to innocuous, discrete communication electronics to sophisticated embedded microprocessors with full access to the Internet. For the purposes of this book, we will focus on security from the data collection point (for example, sensors and devices) to the database and all nodes in between. As you will see, a little security prevention can go a long way to safe guarding your data.

You may also consider security something that needs to be stronger for solutions that are higher risk for humans such as a nuclear power plant or a medical facility. While those are indeed solutions for which we would expect very good security, consider the case of home automation. What would happen if someone were able to hack into your smart home and be able to lock and unlock the doors? In fact, a recent popular baby monitor was found to be easy to hack, allowing hackers to view the images, listen in on conversations, and even manipulate the camera.

You may wonder how someone could use mundane data for nefarious activities. Consider a case where a family who owns a smart home decides to go on vacation. Let's also consider the family is security

---

[17]Not a true fix, but it certainly lowers risk.

[18]I utterly loathe portmanteau (https://en.wikipedia.org/wiki/Portmanteau). Why can't we just say "information and entertainment"?.

[19]http://wired.com/2015/07/hackers-remotely-kill-jeep-highway/.

conscience and has not broadcasted their vacation plans via social media.[20] Let's also assume the only vulnerability hackers can find is a dump of the data from the smart meter on the home. So what? Well, consider that when the family goes on vacation, they use less electricity. Air-conditioning units may be set to higher (lower) temperatures to conserve power, no televisions will be on, no hot water will be used, no cooking is being done, and so on. Thus, the sudden drop in kilowatts used can tell thieves that the family isn't at home. Thus, even innocuous data can be exploited.

■ **Tip**   There is no golden rule or silver bullet for security. Security practices must be constantly adjusted, new mechanisms need to be invented, and you must be generally proactive to keep ahead of those who would circumvent security. That said, you must take security seriously and develop your solution around solid best practices.

Now that I've scared you, let's talk a bit about security for IOT solutions starting with an overview of the most common security threats and how you can handle them. Again, we are examining these so we can prepare to build in security from the ground up in our IOT solutions.

Common Security Threats

Almost every aspect of an IOT solution is at risk for security. You've already seen how easy it would be for someone to exploit IOT devices. Even IOT devices that have security built in may not be sufficient. For example, a recent study from HP[21] showed 8 out of 10 devices failed to implement strong password requirements for access. Indeed, most used something as simple as "1234." As we've discussed, password security is just one area where security needs to be improved.

The report also concluded that, of the devices tested, 60 percent of those that had some form of user interface were vulnerable to attack, 70 percent used unencrypted network services, 80 percent failed to require passwords at all (even their cloud and mobile components), and 90 percent collected some form of personal information or data. With this in mind, the following sections discuss a few key areas we IOT developers need to consider when planning our IOT solutions.

Communication Protocols

The network or communication protocols used can be intercepted, especially if the data is transmitted using well-defined, formatted, clear-text chunks of data (called a *packet* in some protocols). It isn't all that difficult to sense the electrical current on a network cable or intercept a WiFi signal to determine what data is being exchanged. One way to combat this is to use encryption.

Data encryption, while somewhat ubiquitous, is a good way to protect your data. This is especially true if you use encryption that uses 128-bit algorithms and keys that are difficult to guess. Fortunately, encryption has been built into several forms of integrated circuits, making it possible to add it to small electronics. Indeed, you can buy a shield for an Arduino that has encryption functions (`http://sparkfun.com/products/13183`).

---

[20]You don't do this, do you? If you do, stop it! Post those photos after you get back, not while you're neck deep in sand 3,000 miles away.
[21]`www8.hp.com/h20195/V2/GetPDF.aspx/4AA5-4759ENW.pdf`.

Whether you use encryption or not, securing your communication protocols from direct access is another area where some solutions fail. That is, don't run your Ethernet cable outside of your home where it can easily be reached. If you must run Ethernet or similar cables, be sure to place them in conduit buried so that no one can accidentally discover them. If you cannot hide or secure the cabling, paint any exposed cabling the same color as the surrounding area to make it harder to find. A white cable against a white fence post is hard to see if you don't know it is there.

Privacy Policies

One security aspect that is often overlooked is the privacy policy for data collected and retained (sometimes called a *data retention policy*). If you are developing an IOT solution for yourself and storing data on your own database server, there may not be an issue. However, if you are using IOT cloud services, you may want to consider the privacy policy of the service. For example, if you use a service to store data for later access or analysis and decide to cancel the service, what does the company do with the data? Do they leave it where it is for anyone to stumble upon, or does the company delete it after your account expires?

The privacy policy may not be an issue for data that has little or no value and cannot be used against you. But for data such as your address, name, medical history, and so on, the privacy policy could be an issue. Thus, you should always check the privacy and data retention policies of all the services you plan to use.

Remote Maintenance

Companies that provide IOT devices and solutions that have embedded software often provide a means to update the firmware or software periodically. Indeed, it is important to consider solutions that provide this so that you can get the latest fixes and improvements. Not only do you get new features, but more importantly, you can get the latest security updates. For example, Jeep has patched its infotainment systems and provides patches (dealer installation required) to improve security.

However, the mechanism for how the patch or fix is transmitted and applied should be secure. For example, if the patch requires a special administration account, be sure the account is secured with a password that you set. In other words, don't use the factor default—ever. Furthermore, how the patch gets to your system is another concern. If you have to expose your solution to the Internet either to a machine or to a human, you may want to reconsider. Only use patch transmission mechanisms that are secure. That is, downloading the patch to a USB drive and then transferring it to the IOT system and applying it deliberately is more secure than allowing the IOT vendor to automatically update your system.

Password Policies

I discussed password policies previously. In review, be sure to use passwords on all accounts wherever possible and choose your passwords so that they are sufficiently complex enough (not 1234 or the name of your dog, street, or spouse) yet not so much you cannot remember them. Don't ever use default passwords, user accounts without passwords, or the same password for multiple accounts.

Physical Security

I mentioned this topic earlier too. For IOT solutions, this applies to all devices in the system. For those that must be physically outside of a secure area, be sure to make them as secure as possible by minimizing the hardware exposed and locking the enclosure. For example, a camera that sends video to the IOT server can be split so that only the camera portion is external to the building (for instance) and the communication electronics are inside the building. While it is possible for someone to hack the camera (or destroy it), it is less likely they can do anything more than intercept the signal.

Similarly, locking the IOT device in an enclosure can reduce the risk, but the risk is balanced against how sturdy the lock and enclosure is. That is, if the enclosure is made from a material that can be cut or the lock can be easily removed, the security measure only slows down the perpetrator. The determined will still prevail.[22]

Software and Firmware

Another area of concern is the firmware and software (operating systems) used in the IOT solution. We need to ensure the base operating system and other software is secure. More specifically, the software uses secured accounts, cannot be compromised remotely, uses encryption, and can be made hardened from attack. For example, a secure Linux operating system is preferred over an open access system. This also applies to any IOT services outside of the firewall (publically accessible) including web servers, IOT cloud services, and so on.
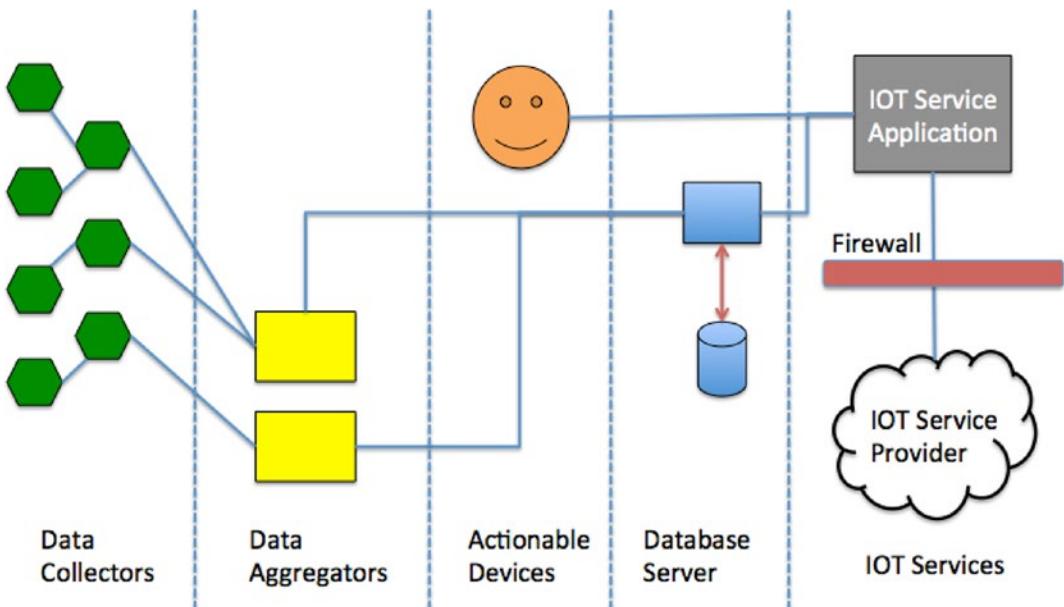
Now that you have seen some of the more common and more serious security risks, let's discuss security from the aspect of an IOT solution.

Securing IOT Solutions

Let's turn our attention to how we can employ security practices for IOT solutions. While this section is not complete in the sense it covers all possible security practices, it is intended to get you thinking about how your own IOT solution should be protected. But before we get to that, let's review a general architecture and nomenclature for IOT solutions. The following describes several types of nodes that you may use to build your IOT solution starting from the lowest layer (the IOT devices) to the highest layer (IOT cloud services). Keep in mind the layers are also ordered from simple to complex regarding capabilities. This happens to also correspond roughly with security at each level. That is, the lower layers are easier to secure (with some exceptions like physical access of external sensors) than the higher layers. Figure 1-10 shows how the nodes could be arranged in an IOT architecture.

---

[22]Which is kind of like windows. Sure, we all lock our windows, but a brick or reasonably sized stone will make short work of the glass.

Figure 1-10. *IOT device architecture*

■ **Tip** I use these terms throughout the book.

- *Data collector*: A sensor, IOT device, and so on, that produces data from some event or observation.

- *Data aggregator*: A node (embedded controller, microcontroller, small computer, and so on) that receives information from one or more data collectors. Its purpose is to aggregate and augment the data for storage at the next layer.

- *Actionable device*: An IOT device that provides some user-controllable feature such as moving a sensor, operating locks, and so on.

- *Database server*: A node, typically a server that stores the data collected for later retrieval and analysis.

- *IOT services*: A computer system that provides an access layer to the database server and actionable devices. It may be systems located inside or outside the solution firewall. Systems are typically Internet servers or cloud services that allow users to view the data and manipulate the actionable devices.

Now let's discuss how to secure each of these layers.

Securing Data Collectors

The data collectors (or IOT devices) are those devices that have one or more sensors that produce data. These are often built with low-cost electronics that provide only the bare minimal capabilities for transmitting the data. I've already suggested using XBee modules to transmit the data via a simplistic protocol. I have also mentioned physically securing the device from tampering. In addition to these principles, you should consider making the data collectors from low-cost electronics, avoiding the temptation to use a more sophisticated small computer when a microcontroller or XBee module (for example) will do. If planned well, devices at this level do not need accounts or other login features.

Securing Data Aggregators

The data aggregator represents the next step in sophistication. Here we need more powerful electronics to manipulate the data. For example, we may want to qualify the data like we did in the plant-monitoring example earlier. Also, data aggregators are the first layer to start using more sophisticated communication protocols such as Ethernet or WiFi. Thus, you need to consider account access security (passwords) as well as remote access capabilities. For example, your data aggregators may support network access via remote logins. These need to be secured in the same manner as any other computing system on the network.

Securing Actionable Devices

Actionable devices can be a bit more challenging to secure. This is because unless you built it yourself, chances are the device is a commercially available device that has more features than you need. You should consider disabling any feature you do not need, ensuring any remote access is secured using as much as possible. Furthermore, I recommend placing the device behind the firewall or an IOT service such as a computer application or system that has a more secure access mechanism. For example, use an application that can be secured using encryption and highly secure remote access to send commands to the device. If you do this, not only do you make it more secure, you can also limit the features or actions available to the outside (Internet).

Securing the Database Server

The database server (if employed) should be secured from access like any other computer system. There are many texts (books, blogs, wikis, and so on) that cover this topic in great detail. I will present some of the best practices in a later chapter. In the interim, consider making your database server a single node in the solution secured from access outside the firewall, all accounts secured with passwords, and the data secured from local file access. In other words, harden your database server.

■ **Tip**   You can find MySQL security best practices at http://mysql.com/why-mysql/presentations/mysql-security-best-practices/.

Securing IOT Services

This layer is the hardest to secure. If you purchase (lease) IOT services from a third party, security is built for you. It is up to you to ensure you use all possible, reasonable practices to secure your data in the service. For example, use good password policies. On the other hand, if you build your own Internet-facing services, you should treat the system with the most secure practices possible. That is, your server should not be accessible remotely without encryption, secure logins, and so on. If you plan to do this, you should consider becoming proficient at securing web services.

Summary

The Internet of Things is an exciting new world for everyone. Those of us young in heart but old enough to remember *The Jetsons* TV series recall seeing a taste of what is possible in make-believe land. Talking toasters, flying cars that spring from briefcases, and robots with attitude notwithstanding, television fantasy of decades ago is coming true. We have wristwatches that double as phones and video players, we can unlock our cars from around the world, we can find out whether our dog has gone outside, and we can even answer the door from across the city. All of this is possible and working today with the advent of the IOT.

In this chapter, you discovered what the IOT is and how IOT solutions are constructed, were introduced to some terminology to describe the architecture of IOT solutions, and saw some examples of well-known IOT solutions. We also discussed the two most critical issues of IOT solutions: big data and security through practical examples and discussion of the high points of the issues. You even saw a bit of source code along the way!

In the next chapter, you will see a number of hardware you can use to build IOT solutions. You will see devices for hosting or reading sensors and devices for collecting, augmenting, storing, and presenting data.

CHAPTER 2

■ ■ ■

Hardware for IOT Solutions

Most IOT solutions, whether they are built by hand or mass produced for commercial sale, are prototyped from basic designs using discrete components. Most hobbyists and enthusiasts base their solutions on components they can buy from retailers (or wholesalers if they produce many units for sale). Much of the commodity hardware therefore is readily available.

This chapter introduces several examples of the more popular commodity hardware you can use in building your IOT solutions. Since there are so many varieties of everything available, I will not attempt or claim this chapter is all-inclusive. For example, I briefly describe a few of the common low-cost computing boards available, but there are more being added every day. It would require a tome several times the size of this book to list all of them. However, I show examples of how to use some of the hardware mentioned in this chapter in Chapter 8.

Included in this chapter are a discussion of popular Arduino microcontroller boards, a brief tutorial on using the Arduino software, a discussion of popular low-cost computing boards, a survey of communication hardware, and even a short discussion on what sort of sensors are available.

While you do not need to become an expert on any of the hardware listed, reading through this chapter will get you the knowledge you need to select which hardware to buy and get started using the hardware. After all, this is a book on using MySQL for IOT solutions rather than an in-depth instruction on how to build IOT solutions—but I have some of this information in this chapter and more in Chapter 8.

Let's begin by examining one of the most common and perhaps the single most versatile hardware component available for IOT solutions—the microcontroller.

HOBBYIST OR ENTHUSIAST: WHAT'S THE DIFFERENCE?

Enthusiasts build things because they can and often because they get a lot of enjoyment out of the build, not necessarily because they need to build it. Enthusiasts also tend to indulge in stockpiling or collecting. A hobbyist builds things for a singular purpose and do not normally indulge as much as enthusiasts. Thus, an enthusiast is a hobbyist who has turned the corner on avocation versus obsession.

Microcontrollers

Microcontrollers[1] are small integrated circuit (IC) chips arranged on a small printed circuit board (PCB) with additional components to support features such as access to pins on the chip for connecting other components such as LEDs, servos, sensors, and more. Microcontrollers tend to be limited in the amount of

---

[1] https://en.wikipedia.org/wiki/Microcontroller

memory available, have limited command features (such as operations for interacting with the hardware), have few connections (typically a programming and power connector), and often use specialized programming languages.

There are many microcontrollers available ranging from a bare IC with the AVR firmware installed[2] to a microcontroller-based platform that supports loadable programs and a wide range of hardware expansion. Fortunately, many such platforms are available.

One of the most popular and widely available microcontroller platforms is the Arduino. The following sections present a wide array of information about the Arduino including examples of boards you can buy and even a tutorial on how to program the Arduino.

■ **Note**   I often use the terms *microcontroller* or *microcontroller platform* to discuss a product category and *board* to refer to a specific version of the platform.

What Is an Arduino?

The Arduino is an open source hardware prototyping platform supported by an open source software environment. It was first introduced in 2005 and was designed with the goal of making the hardware and software easy to use and available to the widest audience possible. Thus, you don't have to be an electronics expert to use the Arduino.

The original target audience included artists and hobbyists who needed a microcontroller to make their designs and creations more interesting. However, given its ease of use and versatility, the Arduino has quickly become the choice for a wider audience and a wider variety of projects.

This means you can use the Arduino for all manner of projects from reacting to environmental conditions to controlling complex robotic functions. The Arduino has also made learning electronics easier through practical applications.

Another aspect that has helped the rapid adoption of the Arduino platform is the growing community of contributors to a wealth of information made available through the official Arduino web site (http://arduino.cc/en/). When you visit the web site, you will find an excellent "getting started" tutorial as well as a list of helpful project ideas and a full reference guide to the C-like language for writing the code to control the Arduino (called a *sketch*).

The Arduino also provides an integrated development environment called the Arduino IDE. The IDE runs on your computer (called the *host*), where you can write and compile sketches and then upload them to the Arduino via USB connections. The IDE is available for Linux, Mac, and Windows. It's designed around a text editor that is especially designed for writing code and a set of limited functions designed to support compiling and loading sketches.

Sketches are written in a special format consisting of only two required methods—one that executes when the Arduino is reset or powered on and another that executes continuously. Thus, your initialization code goes in setup(), and your code to control the Arduino goes in loop(). The language is C-like, and you may define your own variables and functions. For a complete guide to writing sketches, see http://arduino.cc/en/Tutorial/Sketch.

You can expand the functionality of sketches and provide for reuse by writing libraries that encapsulate certain features such as networking, using memory cards, connecting to databases, doing mathematics, and the like. Many such libraries are included with the IDE. There are also some libraries written by others and contributed to Arduino.cc through open source agreements—some of which have been bundled with the IDE.

---

[2]http://atmel.com/images/doc8161.pdf

The Arduino supports a number of analog and digital *pins* that you can use to connect to various devices and components and interact with them. The mainstream boards have specific pin layouts, or *headers*, that allow the use of expansion boards called *shields*. Shields let you add additional hardware capabilities such as Ethernet, Bluetooth, and XBee support to your Arduino. The physical layout of the Arduino and the shield allow you to stack shields. Thus, you can have an Ethernet shield as well as an XBee shield, because each uses different I/O pins. You will learn about the use of the pins and shields as you explore how to apply the Arduino to IOT networks.

The Arduino is used best with a breadboard when developing or prototyping circuits. A breadboard is designed to allow you to plug in your electrical components and provide interconnectivity in columns so that you can plug the leads of two components into the same column and therefore make a connection. The board is split into two rows, making it easy to use an IC in the center of the board. Wires (called *jumpers*) can be used to connect the circuit on the breadboard to the Arduino. You will see an example of this later in this chapter.

The next sections examine some of the various Arduino boards—both Arduino branded and third party. Many more boards and variants are available, and a few new ones are likely to be out by the time this book is printed, but these are the ones that I use in my IOT projects and experiments. Any of these can be an excellent basis for your own projects.

Arduino Models

A growing number of Arduino boards are available. Some are configured for special applications, whereas others are designed with different processors and memory configurations. Some boards are considered official Arduino boards because they're branded and endorsed by Arduino.cc. Because the Arduino is open source and, more specifically, licensed using a Creative Commons Attribution Share-Alike license, anyone can build Arduino-compatible boards (often called Arduino *clones*). However, you must follow the rules and guidelines set forth by Arduino.cc.[3] This section examines some of the more popular Arduino-branded boards.
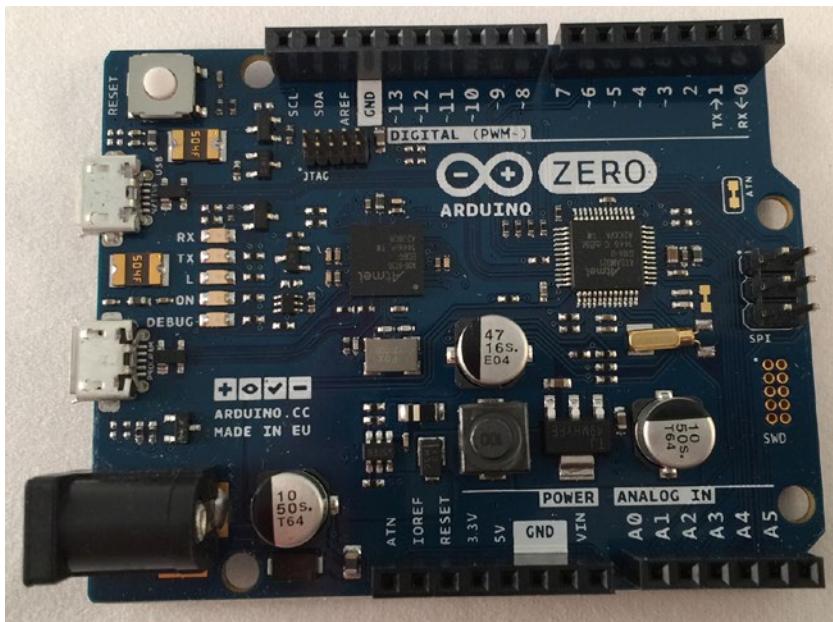
The basic layout of an Arduino board consists of a USB connection, a power connector, a reset switch, LEDs for power and serial communication, and a standard-spaced set of headers for attaching shields. The official boards sport a distinctive blue PCB with white lettering. With the exception of one model, all the official boards can be mounted in a chassis (they have holes in the PCB for mounting screws). The exception is an Arduino designed for mounting on a breadboard.

Arduino Zero

The Arduino Zero is the latest small-footprint Arduino board available from Arduino.cc. It has the same physical layout as the boards in the Arduino Uno family (`http://arduino.cc/en/Main/ArduinoBoardUno`) but has a much faster, 32-bit processor. The board is similar to the Leonardo board (see the "Arduino Leonardo" section) with 20 digital I/O pins, of which 18 can be used as analog pins. It has more memory with 256KB of flash memory and 32KB of SRAM. The clock speed is also faster at 48MHz. Figure 2-1 shows an early release of the Arduino Zero board.

---

[3]For a complete description of the Arduino.cc license policies and more information about building and selling your own Arduino-compatible board, see `http://arduino.cc/en/Main/FAQ`.

Figure 2-1. *Arduino Zero*

The board comes equipped with a USB host port so you can write sketches to access devices via USB such as a keyboard or mouse.

The Arduino Zero is an excellent choice for most Arduino projects and can solve a number of issues with memory and performance using the older boards. While not a replacement for poor or inefficient programming, the Arduino Zero can improve solutions that require more processing power and, more importantly, more memory.

■ **Tip** The Arduino Zero runs on 3.3V. Take extra care when using shields and connecting components to ensure you do not damage the board.
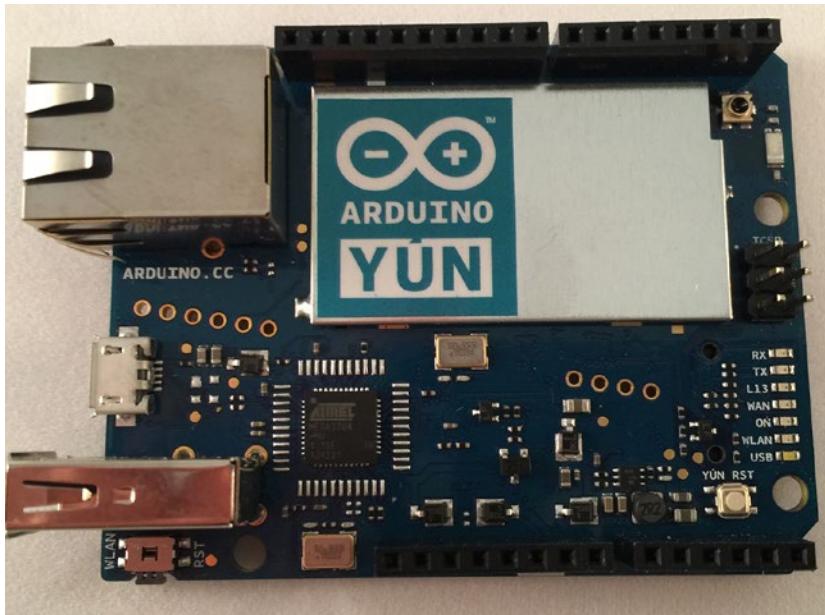
While this is the newest Arduino board in my portfolio of microcontrollers, I have used this board to test sketches that require more memory. Indeed, it has already proven to be an excellent choice for a node in my IOT solutions where I aggregate or augment the data. The added memory means I can also issue queries on the database server for looking up values for calculations.

You can find specific documentation for the Arduino Zero at http://arduino.cc/en/Main/ArduinoBoardZero.

Arduino Yún

The Arduino Yún is a very different Arduino board. While it is and supports use as a normal Arduino board (you can run the same sketches), the Yún has two processors: the Atmel ATmega32U4 is a Leonardo-compatible microcontroller, and the Atheros AR9331 runs a scaled-down version of Linux and the OpenWrt wireless stack.

The Yún has a USB host controller as well as both WiFi and Ethernet networking. Indeed, you can use the WiFi capabilities to form a wireless access port, making it possible for IOT solutions to host their own WiFi-connected devices. Like the Zero, this board runs on 3.3V, requiring you to select your components and shields carefully to ensure you do not damage the board. Figure 2-2 depicts the Arduino Yún.

Figure 2-2. *Arduino Yún*

The Yún is not a board for everyone. However, if you need to be able to combine the use of Arduino shields with more powerful scripts written in Python, the Yún is perhaps a good choice for combining the power of Python with the versatility of Arduino-compatible components and shields. There are other boards that do this a bit better, but most do not include the WiFi access port capability. If you need or want to provide a powerful wireless access port feature in your solution that can run any of your Arduino sketches, the Yún is a good choice.
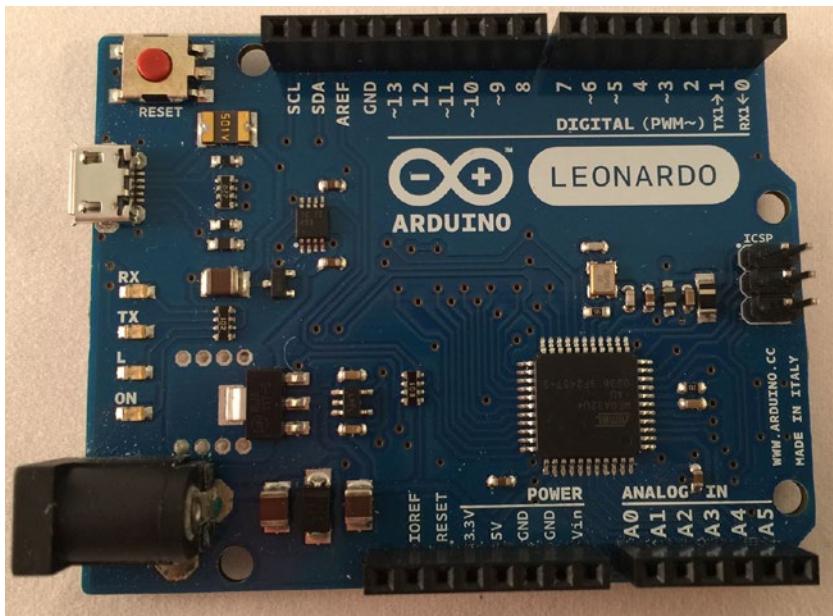
So far, I've used my Yún to create discrete WiFi networks for forensics and diagnosis of networking problems. Using the Yún as an access port permits me to do things I would not do on my own WiFi network. After all, if things go bad, I can just reset my Yún and try it again. It is far more risky to try that with your own wireless access router.

You can find specific documentation for the Arduino Yún at `http://arduino.cc/en/Guide/ArduinoYun`.

Arduino Leonardo

The Leonardo board is an evolution of the older, Uno-based boards. Although it supports the standard header layout, it doesn't support some of the older shields. However, it adds a faster processor and a USB controller that allows the board to appear as a USB device to the host computer. The newer ATmega32u4 processor has 20 digital I/O pins, of which 12 can be used as analog pins and 7 can be used as a pulse-width modulation (PWM) output. It has 32KB of flash memory and 2.5KB of SRAM.

The Leonardo has more digital pins than its predecessor but continues to support most shields. The USB connection uses a smaller USB connector. The board is also available with and without headers. Figure 2-3 depicts an official Leonardo board.

Figure 2-3. *Arduino Leonardo*

I have used the Leonardo to upgrade many of my older Arduino-based projects. While the board doesn't offer a lot more capability (but there is some, particularly memory) than some of the older boards, I am able to use the newer shields, which are starting to require the new header layout.
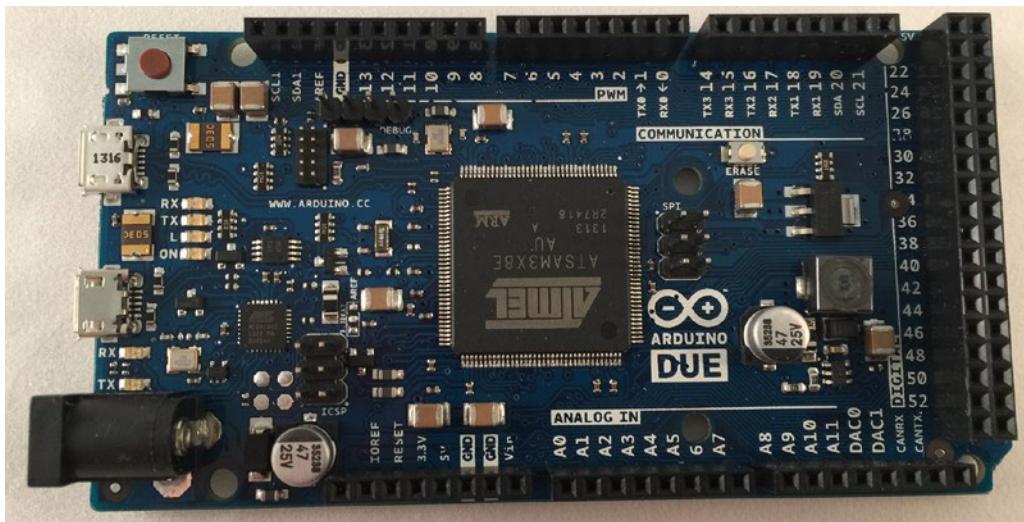
You can find specific documentation for the Arduino Leonardo at http://arduino.cc/en/Main/ArduinoBoardLeonardo.

Arduino Due

The Arduino Due is a new, larger, faster board based on the Atmel SAM3X8E ARM Cortex-M3 processor. The processor is a 32-bit processor, and the board supports a massive 54 digital I/O ports, of which 14 can be used for PWM output; 12 analog inputs; and 4 UART chips (serial ports); as well as two digital-to-analog (DAC) and two two-wire interface (TWI) pins. The new processor offers several advantages.

- 32-bit registers

- DMA controller (allows CPU-independent memory tasks)

- 512KB flash memory

- 96KB SRAM

- 84MHz clock

The Due has a larger form factor (called the *mega footprint*) but still supports the use of standard shields, but it also supports the mega format shields. The new board has one distinct limitation: unlike other boards that can accept up to 5V on the I/O pins, the Due is limited to 3.3V on the I/O pins. Figure 2-4 shows an Arduino Due board.
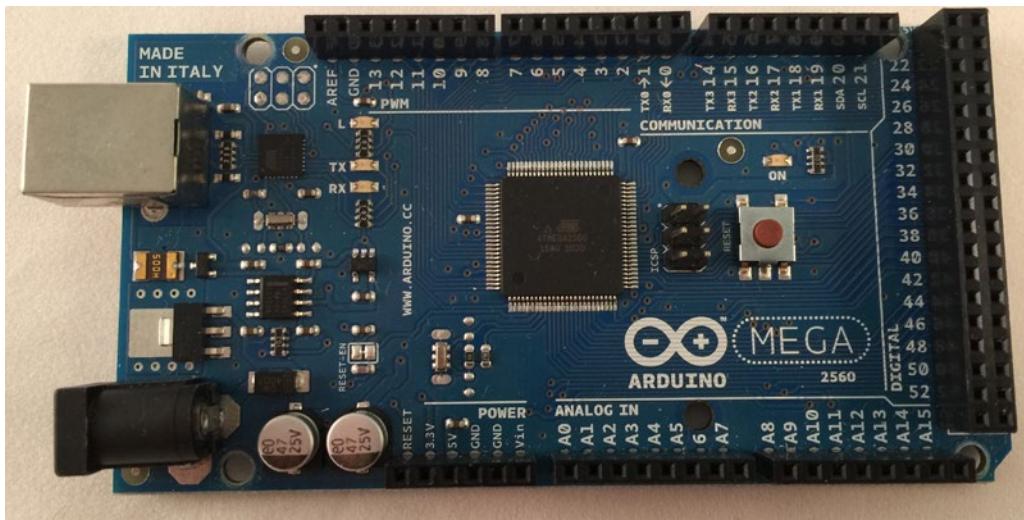
Figure 2-4. *Arduino Due*

The Arduino Due can be used for projects that require more processing power, more memory, and more I/O pins. I've used the Due in rare cases where I have needed a lot more I/O pins or more memory. However, with the new Zero, unless I need the added I/O pins, I'd likely use the Zero where physical size is an issue. If you don't have a space issue, look to the Due for your projects that require the maximum hardware performance.

You can find specific documentation for the Arduino Due at `http://arduino.cc/en/Main/ArduinoBoardDue`.

Arduino Mega 2560

The Arduino Mega 2560 is an older form of the Due. It's based on the ATmega2560 processor (hence the name). Like the Due, the board supports a massive 54 digital I/O ports, of which 14 can be used as PWM output; 16 analog inputs; and 4 UARTs (hardware serial ports). It uses a 16MHz clock and has 256KB of flash memory. Figure 2-5 shows the Arduino Mega 2560 board.

Figure 2-5. *Arduino Mega 2560*

The Mega 2560 is essentially a larger form of the standard Arduino (Uno, Duemilanove, and so on) and supports the standard shields. Interestingly, the Arduino Mega 256 is the board of choice for Prusa Mendel and similar 3D printers that require the use of a controller board named RepRap Arduino Mega Pololu Shield (RAMPS). Indeed, this is where most of my Mega 2560 boards are right now—in my 3D printers! That said, like the Due, you could use this board if you require additional I/O pins or more memory and size is not an issue (it is the same size and the Due).

You can find specific documentation for the Arduino Due at `http://arduino.cc/en/Main/ArduinoBoardDue`.
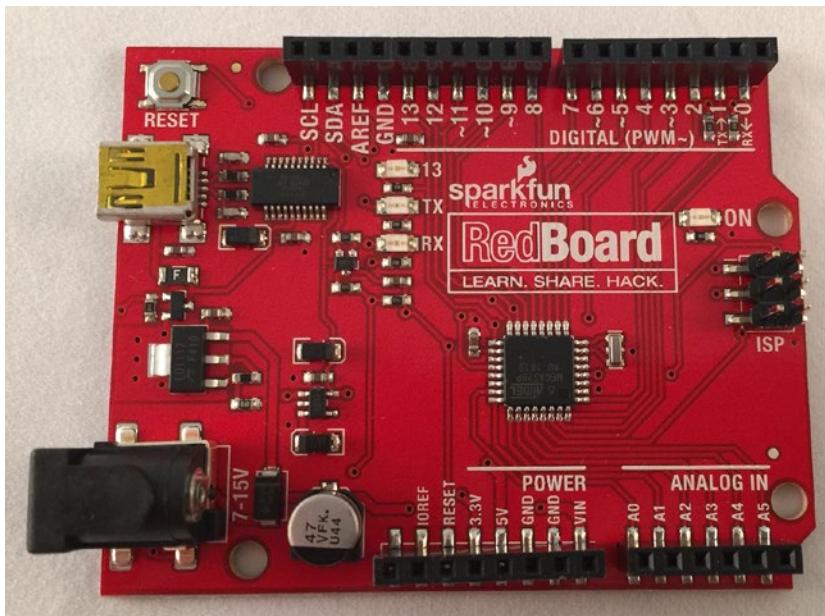
Arduino Clones

There are a growing number of third-party Arduino boards (also called *clones*). They range in form factor, components, and even features. Fortunately, most are 100 percent Arduino compatible, making it easy to use them in your Arduino-based projects. Even if the board is not 100 percent compatible with a particular Arduino board (for example, the Leonardo), third-party boards have been added to the Arduino integrated development environment requiring only that you select the specific board when compiling. You will see this later in this chapter.

The following sections describe some of the third-party Arduino boards that I use in my projects. Chances are you will encounter some of these and even more like them. As you will see, some have very different layouts, giving you more options when implementing your projects. Since these boards retain compatibility with the Arduino-branded boards, I describe them briefly by focusing on the unique features.

Sparkfun Redboard

The Sparkfun Redboard (`http://sparkfun.com/products/12757`) is an advanced version of the Arduino Uno. It uses the same boot loader, the addition of an FTDI interface (used in the older Duemilanove boards), and the R3 shield compatibility of the latest Arduino UNO R3. In effect, Sparkfun has reinvented the Arduino Uno by bringing back the best qualities of the older boards. And, yes, it is red. Figure 2-6 shows the Sparkfun Redboard.

Figure 2-6. *Sparkfun Redboard*

What I like most about the Redboard is it is a bit cheaper than the Arduino-branded boards and is readily available from Sparkfun. Better still, Sparkfun stocks a number of shields and accessories that complement the Redboard. In fact, Sparkfun has an excellent kit called the Sparkfun Inventor's Kit (`http://sparkfun.com/products/12060`) that help you learn about Arduino programming and building circuits.
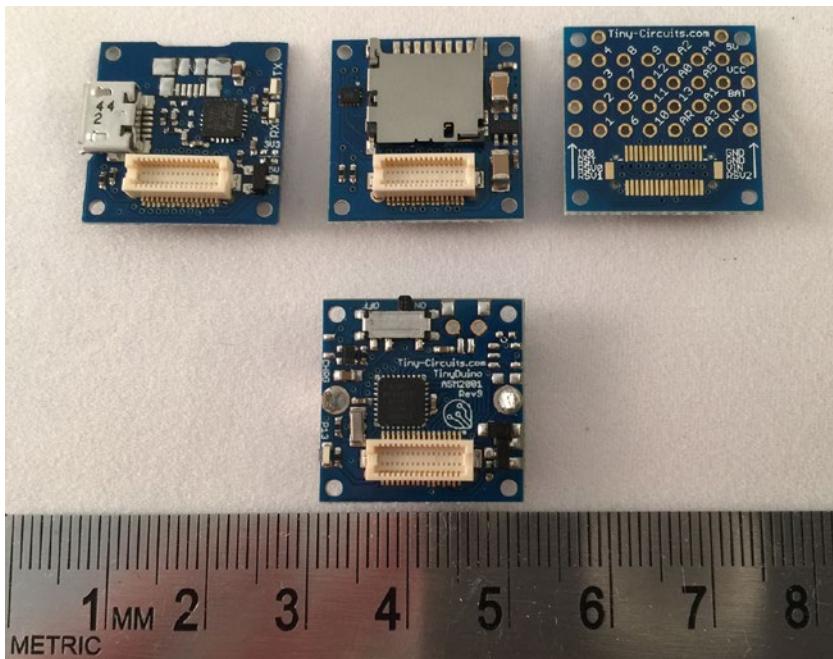
■ **Tip**   If you have friends and family interested in learning more about Arduino, the Sparkfun Inventor's Kit would make an excellent gift.

You can find specific documentation for the Sparkfun Redboard at `https://learn.sparkfun.com/tutorials/redboard-hookup-guide`.

TinyCircuits TinyDuino

The TinyDuino is an interesting diversion from the standard Uno physical layout. As the name suggests, it is a small board. In fact, it is only slightly larger than a U.S. quarter![4] As you can see in Figure 2-7, the main board (center bottom) is about 21×21mm. That's small.

---

[4]Or as my grandmother used to say, "A twenty-and-five-cent piece."

Figure 2-7. *TinyDuino*

Figure 2-7 shows four modules (also called *boards* or *core modules*); from left to right on the top row is a USB shield used to program the TinyDuino, a MicroSD card reader, and a prototyping board. The module in the center bottom is the main processor module.
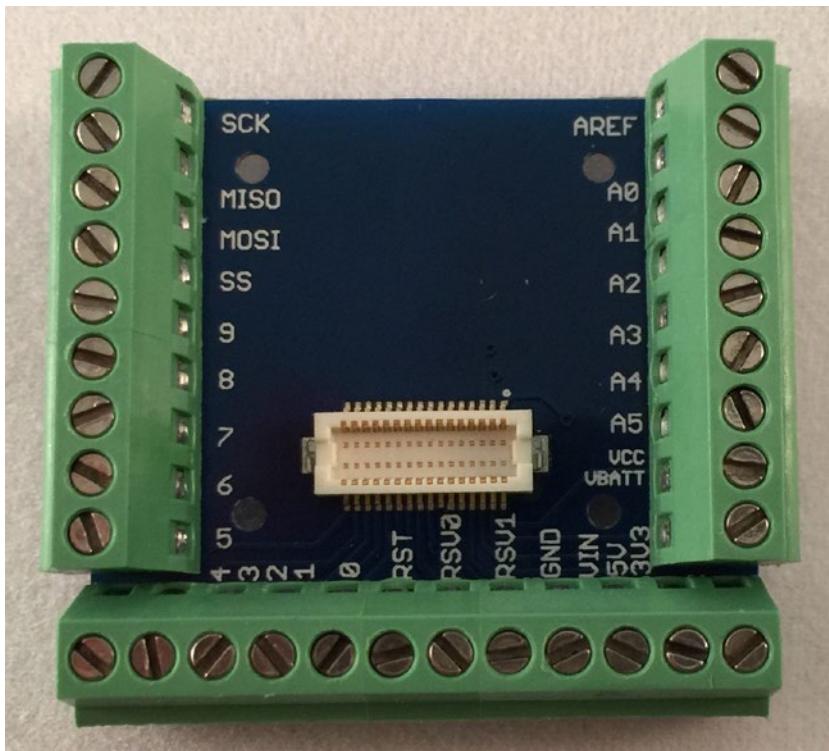
The TinyDuino is compatible with the Arduino Uno with the same processor, memory, pins, and so on. There are some minor differences but nothing that is of any concern. In fact, the TinyDuino will run all of your sketches and can be programmed as if it were an Arduino Uno. You just cannot use Arduino shields with the TinyDuino.

Aside from the small size, the TinyDuino processor board has a battery holder that uses a CR1632 coin cell battery to power the board. This allows you to keep your Arduino solution very small and even run on battery power! Better still, there are additional versions of the main processor board that have a Lithium battery connector and even one without battery support that you can use to hardwire a power source.

The boards connect via a microconnector that permits you to stack them on top of one another. They can be stacked in any order with some exceptions. The processor board does not have a connector on the bottom, and the prototype boards do not have a connector on top. Regardless, even if you stack all four boards as shown, the TinyDuino stack is only about 20mm tall.

■ **Tip**   You must have the USB board in order to program the TinyDuino, but once it's programmed, you can remove the board.

There is one other board that I consider a must-have if you need to use more than a few pins. The prototype boards have a number of pins broken out, but not all of them. The terminal module shown in Figure 2-8 allows access to all the Arduino pins and comes complete with screw terminals for every pin! That's cool. While it is a bit larger than the core processor module, it is still very small.

Figure 2-8. *TinyDuino terminal board*

Better still, TinyCircuits (http://tiny-circuits.com/), the maker of the TinyDuino, offers a growing selection of modules to add to your tiny Arduino kit. There are modules for LEDs, Bluetooth, WiFi, real-time clock (RTC), accelerometer, audio, dual seven-segment display, GPS, and even a micro-sized OLED screen. With so many modules available, you can build a really powerful Arduino solution in a fraction of the size of the normal Arduino boards.

Thus, the TinyDuino represents an excellent choice for solutions that need to be as small as possible. I've seen these boards used in wearables,[5] inside small toys, and even in wristwatch-sized Arduino devices. The small size and lower power make it ideal for hiding in small places.

You can find specific documentation for the TinyDuino at http://tiny-circuits.com/tinyduino_overview.
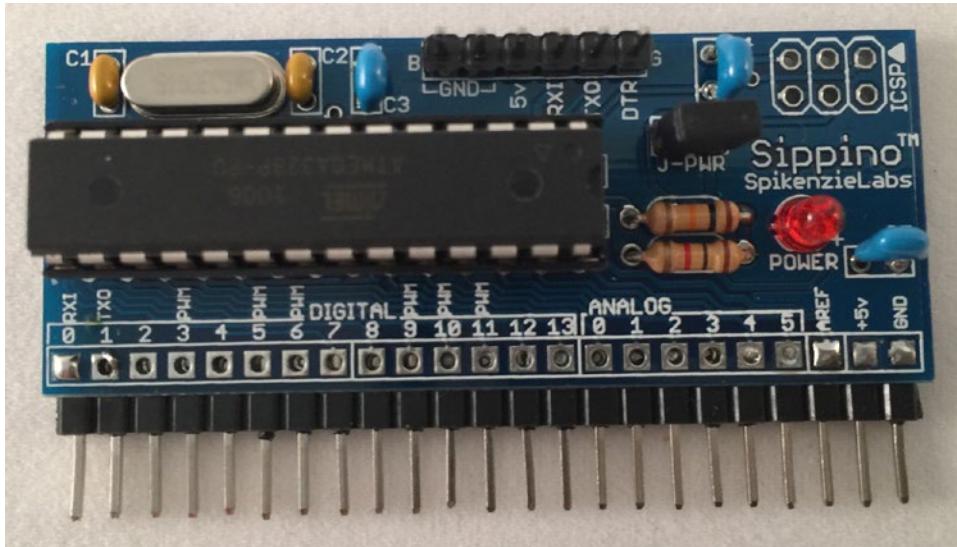
SpikenzieLabs Sippino

The Sippino from SpikenzieLabs (www.spikenzielabs.com) can be used on a solder-less breadboard. It costs less because it has fewer components and a much smaller footprint. Fortunately, SpikenzieLabs also provides a special adapter called a *shield dock* that allows you to use a Sippino with standard Arduino shields.

---

[5]TinyCircuits also makes a LilyPad-compatible module line the size of a U.S. dime (ten-cent piece) called the TinyLily (http://tiny-circuits.com/products/tiny-lily.html).
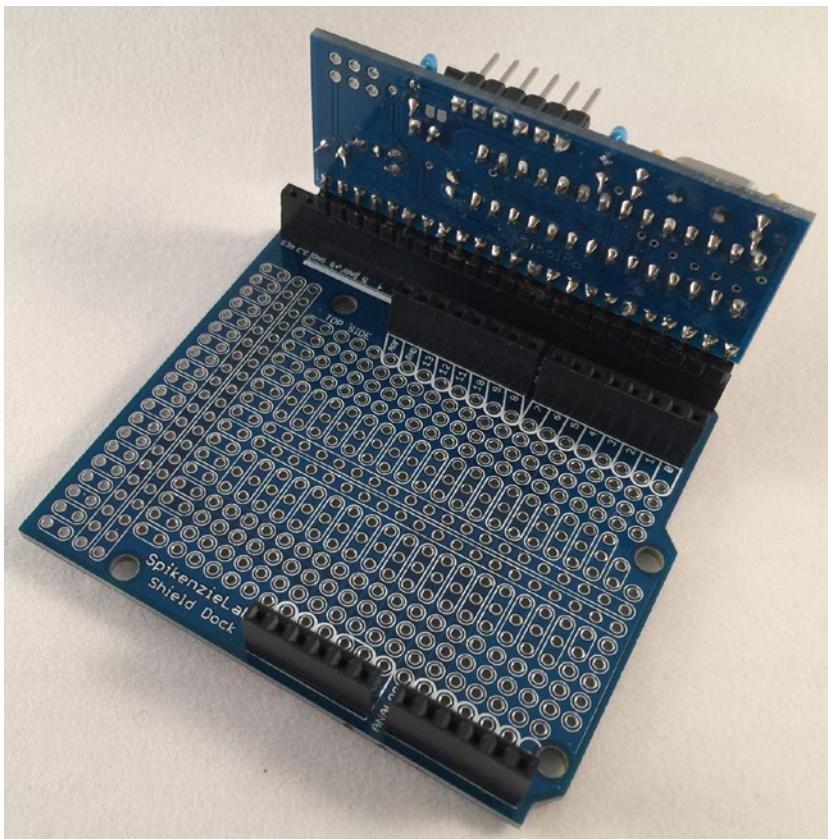
It's based on the ATmega328 processor and has 14 digital I/O pins, of which 6 can be used as PWM output, and 6 analog input pins. The Sippino board has 32KB of flash memory and 2KB of SRAM. Figure 2-9 shows a Sippino with breadboard headers.

Figure 2-9. *Sippino*

The Sippino doesn't have a USB connection, so you have to use an FTDI cable to program it. The good news is you need only one cable no matter how many Sippinos you have in your project. I have a number of Sippinos and use them in many of my Arduino projects where space is at a premium.

The shield dock is an amazing add-on that lets you use the Sippino as if it were a standard Uno or Duemilanove. Figure 2-10 shows a Sippino mounted on a shield dock.

Figure 2-10. *Sippino shield dock*

I have used the Sippino in numerous projects from small geeky alarm clocks (think servos, analog gauges, and old-school glowing bulbs) to sensor nodes in sensor networks. The small size and card layout make it easy to hide the Sippino in a small enclosure. If you need or want something compatible with the older Arduino boards (or need to occasionally use it with an Arduino Uno–compatible shield) in a small package, the Sippino is a good choice.

Another reason I like these boards is they are sold as kits where you assemble the board yourself. The assembly is easy, and except for reading the values of the resistors (a skill every electronics hobbyist must master), the components are easy to align to the correct spot on the board.

You can find specific documentation for the Sippino and Shield dock at http://spikenzielabs.com.
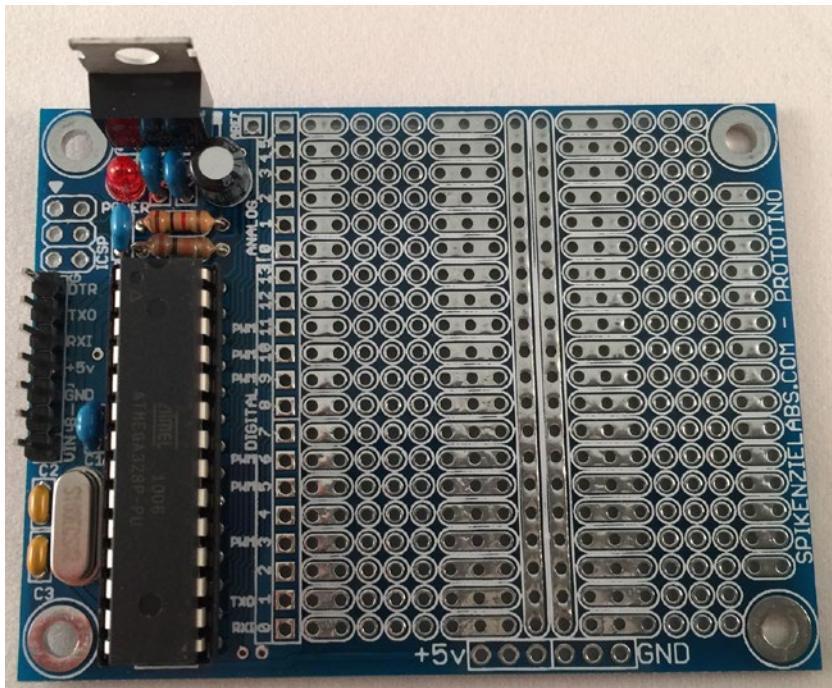
Spikenzie Labs Prototino

The Prototino is another product of SpikenzieLabs. It has the same components as the Sippino, but instead of a breadboard-friendly layout, it's mounted on a PCB that includes a full prototyping area. Like the Sippino, it's based on the ATmega328 processor and has 14 digital I/O pins, of which 6 can be used as PWM output, and 6 analog input pins. The Prototino board has 32KB of flash memory and 2KB of SRAM.

The Prototino is ideal for building solutions that have supporting components and circuitry. In some ways it's similar to the Nano, Mini, and similar boards in that you can use it for permanent installations. But unlike those boards (and even the Arduino Pro), the Prototino provides a space for you to add your

components directly to the board. I have used a number of Prototino boards for projects where I have added the components to the Prototino and installed it in the chassis. This allowed me to create a solution using a single board and even build several copies quickly and easily.

Like the Sippino, the Prototino doesn't have a USB connection, so you have to use an FTDI cable to program it. Figure 2-11 shows a Prototino board.

Figure 2-11. *Prototino*

I've used the Prototino in a number of projects where I had prototyped a circuit on a breadboard and wanted to quickly transfer it to a permanent board.[6] In fact, I started using the Prototino in many of my early projects because it was so easy to work with the breadboard layout of the solder pads.

Like the Sippino, the Prototino is available only as a self-assembly kit. You can find specific documentation for the Prototino at http://spikenzielabs.com.
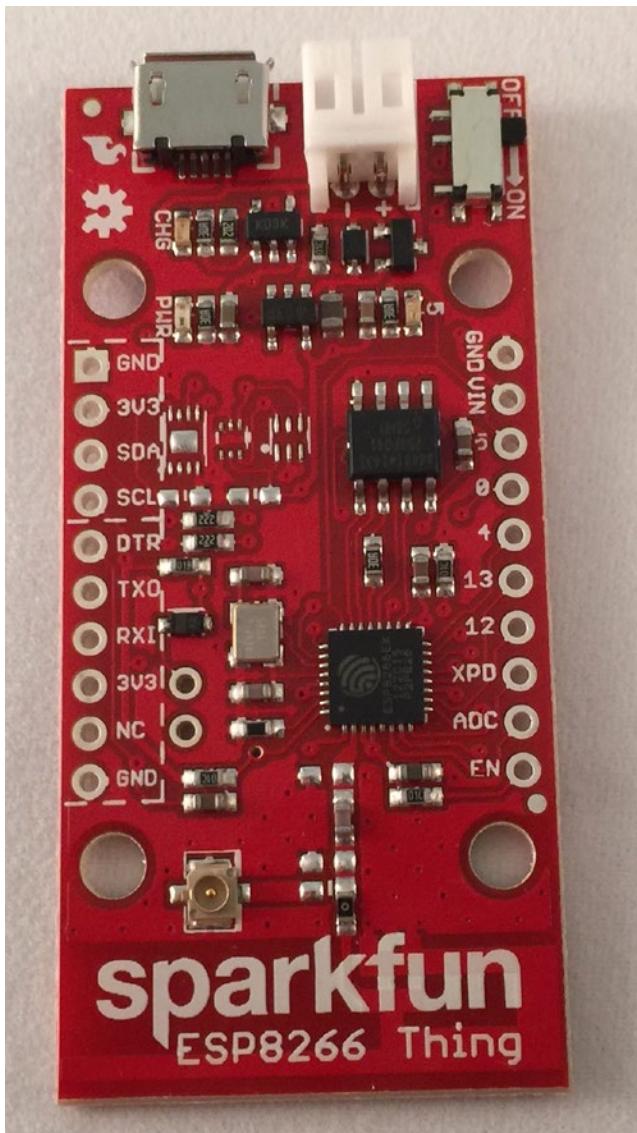
Sparkfun ESP8266 WiFi Module

There is one last board I have in my collection of third-party Arduino boards. The ESP8266 WiFi module is one of the latest additions to the category. It is sometimes listed under the IOT category because of its unique features. Sparkfun's ESP8266 Thing has many cool features such as a LiPo battery connector, on/off switch (very handy), an external antenna connector (also handy for increasing range), a USB programming port, and of course all the available pins broken out. The board is also very small.

---

[6]Well, permanent until you unsolder it!

You can program the ESP8266 with the Arduino integrated development environment by adding a special add-on. See https://github.com/esp8266/Arduino for more details on how to configure the board manager. Figure 2-12 shows the Sparkfun ESP8266 board.

Figure 2-12. *ESP8266 Thing*
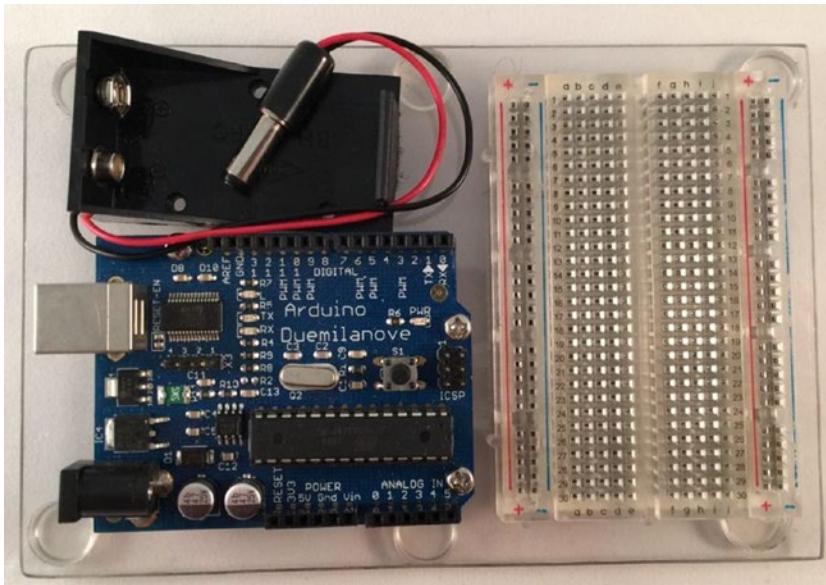
The ESP8266 represents yet another small physical layout board that you can use to create nodes for your IOT network (or single solution). I haven't used mine in any real projects yet, but the power of the module is appealing.

Sparkfun also sells an Arduino-sized board called the ESP8266 WiFi shield that you can use with your Arduino boards (http://sparkfun.com/products/13287). It can be used to provide WiFi capabilities to your Arduino projects. I have one of these too and am using it in place of the normal Arduino WiFi shield (see the "Recommended Accessory" sidebar) mainly because I wanted to experiment with the ESP8266 WiFi capabilities without the extra work to program the ESP8266 Thing.

You can find more information about the ESP8266 in Sparkfun's excellent resource library at https://learn.sparkfun.com/tutorials/esp8266-thing-hookup-guide.

RECOMMENDED ACCESSORY

If you plan to do any prototyping with your Arduino or other microcontroller board, I recommend getting or making your own experiment base (sometimes called a *breadboard holder*). You can easily make one from acrylic as I have done in the example shown here. In this case, I have an older Duemilanove board mounted next to a half-sized breadboard. I even had room to place a 9V battery holder for powering the experiment.

You can also buy them pre-drilled for most Arduino Uno–sized boards. Both Sparkfun (http://sparkfun.com/products/11235) and Adafruit (http://adafruit.com/products/275) stock excellent examples. You can also find these for other boards such as the Raspberry Pi.

Keeping the microcontroller and breadboard together makes it possible to move the experiment from one place to another or stop and resume at a later date.

Arduino Tutorial

This section is a short tutorial on getting started using an Arduino. It covers obtaining and installing the IDE and writing a sample sketch. Rather than duplicate the excellent works that precede this book, I cover the highlights and refer readers who are less familiar with the Arduino to online resources and other books that offer a much deeper introduction. Also, the Arduino IDE has many sample sketches that you can use to explore the Arduino on your own. Most have corresponding tutorials on the `http://arduino.cc` site.

Learning Resources

A lot of information is available about the Arduino platform. If you're just getting started with the Arduino, Apress offers an impressive array of books covering all manner of topics concerning the Arduino, ranging from getting started using the microcontroller to learning the details of its design and implementation. The following is a list of the more popular books:

- *Beginning Arduino* by Michael McRoberts (Apress, 2010)

- *Practical Arduino: Cool Projects for Open Source Hardware (Technology in Action)* by Jonathan Oxer and Hugh Blemings (Apress, 2009)

- *Arduino Internals* by Dale Wheat (Apress, 2011)

There are also some excellent online resources for learning more about the Arduino, the Arduino libraries, and sample projects. The following are some of the best:

- *Arduino.cc*: `http://arduino.cc/en/`

- *Adafruit tutorials*: `http://learn.adafruit.com/`

- *Make tutorials*: `http://makezine.com/category/electronics/arduino/`
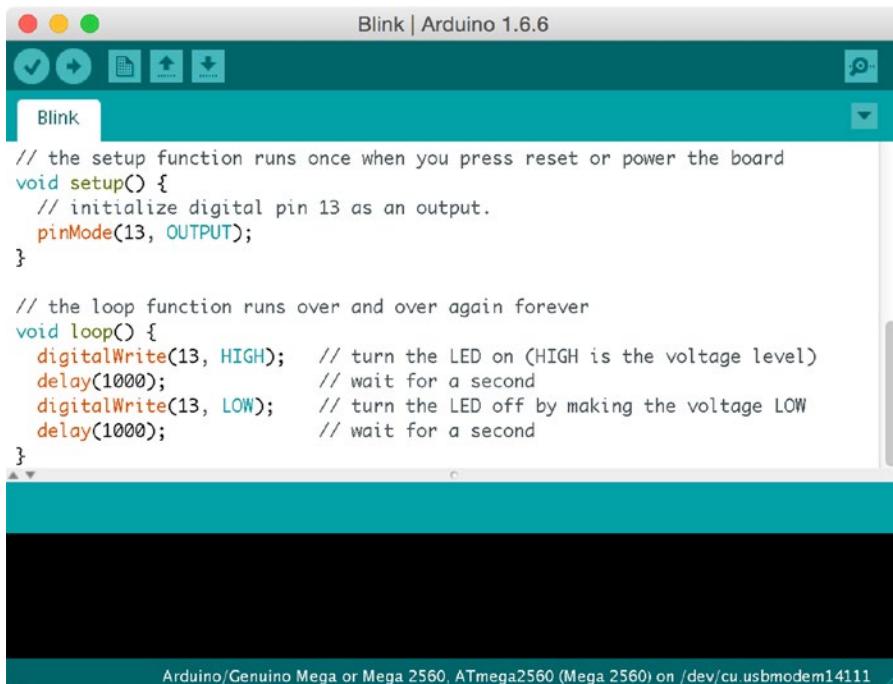
The Arduino IDE

The Arduino IDE is available for download for the Mac, Linux (32- and 64-bit versions), and Windows platforms. You can download the IDE from `http://arduino.cc/en/Main/Software`. There are links for each platform as well as a link to the source code if you need to compile the IDE for a different platform.

Installing the IDE is straightforward. I omit the actual steps of installing the IDE for brevity, but if you require a walk-through of installing the IDE, you can follow the Getting Started link on the download page or read more in *Beginning Arduino* by Michael McRoberts (Apress, 2010).

■ **Tip** See `http://arduino.cc/en/Guide/Howto` if you need help installing the drivers on Windows.

Once the IDE launches, you see a simple interface with a text editor area (a white background by default), a message area beneath the editor (a black background by default), and a simple button bar at the top. The buttons are (from left to right) Compile, Compile and Upload, New, Open, and Save. There is also a button to the right that opens the serial monitor. You use the serial monitor to view messages from the Arduino sent (or printed) via the Serial library. You will see this in action in your first project. Figure 2-13 shows the Arduino IDE.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```
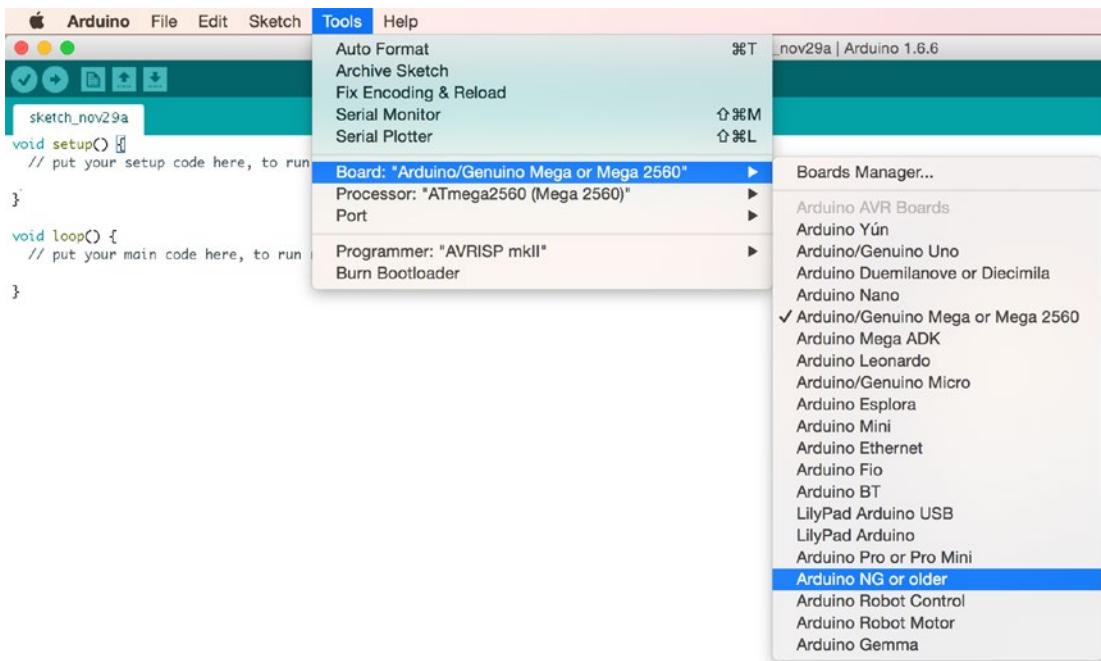
Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) on /dev/cu.usbmodem14111

Figure 2-13. *The Arduino IDE*

Notice that in Figure 2-13 you can see a sample sketch (called a *blink*) and the result of a successful compile operation. Notice also at the bottom it tells you that you're programming an Arduino Uno board on a specific serial port.
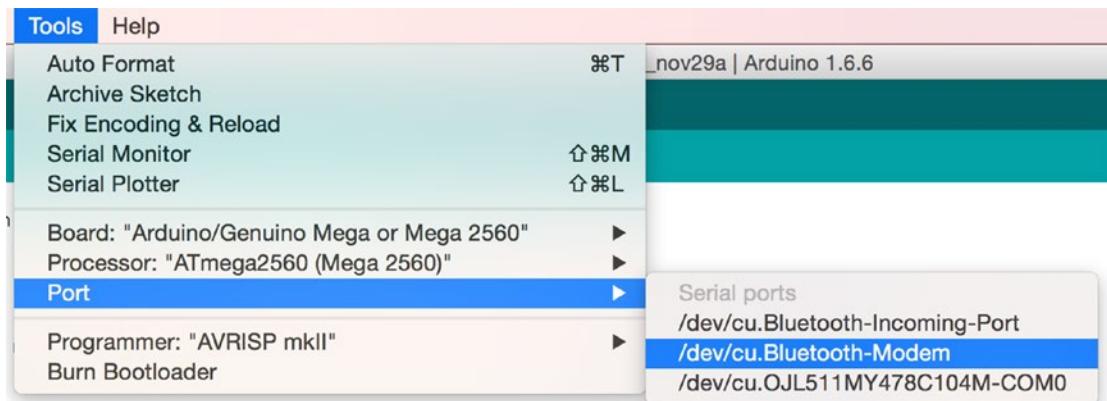
Because of the differences in processor and supporting architecture, there are some differences in how the compiler builds the program (and how the IDE uploads it). Thus, one of the first things you should do when you start the IDE is choose your board from the Tools ➤ Board menu. Figure 2-14 shows a sample of selecting the board on a Mac.

Figure 2-14. *Choosing the Arduino board*

Notice the number of boards available. Be sure to choose the one that matches your board. If you're using a clone board, check the manufacturer's site for the recommended setting to use. If you choose the wrong board, you typically get an error during upload, but it may not be obvious that you've chosen the wrong board. Because I have so many different boards, I've made it a habit to choose the board each time I launch the IDE.

The next thing you need to do is choose the serial port to which the Arduino board is connected. To connect to the board, use the Tools ➤ Port menu option. Figure 2-15 shows an example on a Mac. In this case, no serial ports are listed. This can happen if you haven't plugged your Arduino into the computer's USB ports (or hub), if you had it plugged in but disconnected it at some point, or if you haven't loaded the drivers for the Arduino (Windows). Typically, this can be remedied by simply unplugging the Arduino and plugging it back in and waiting until the computer recognizes the port.

Figure 2-15. *Choosing the serial port*

■ **Note**   If you use a Mac, it doesn't matter which port you choose—either the one that starts with `tty` or the one that starts with `cu` will work.

Now that you have your Arduino IDE installed, you can connect your Arduino and set the board and serial port. You see the LEDs on the Arduino illuminate. This is because the Arduino is getting power from the USB. Thus, you don't need to provide an external power supply when the Arduino is connected to your computer. Next, let's dive into a simple project so you can see the Arduino IDE and learn how basic sketches are built, compiled, and uploaded.
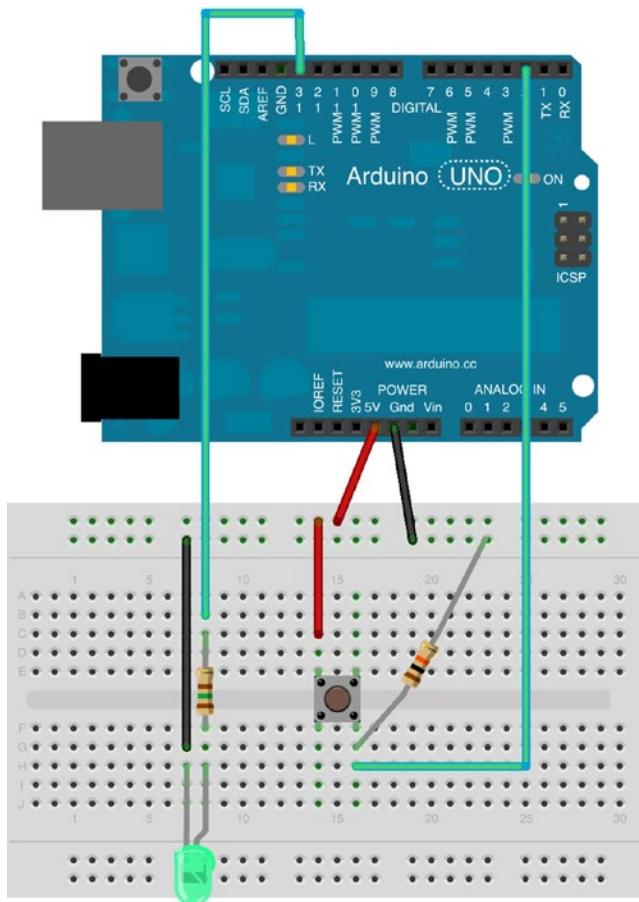
Project: Hardware "Hello, World!"

The ubiquitous "Hello, World!" project for the Arduino is the blinking light. The project uses an LED, a breadboard, and some jumper wires. The Arduino turns on and off through the course of the loop() iteration. That's a fine project for getting started, but it doesn't relate to how sensors could be used.

Thus, in this section, you expand on the blinking light project by adding a sensor. In this case, you still keep things simple by using what is arguably the most basic of sensors: a pushbutton. The goal is to illuminate the LED whenever the button is pushed.

Wiring the Circuit

Let's begin by assembling an Arduino. Be sure to disconnect (power down) the Arduino first. You can use any Arduino variant that has I/O pins. Place one LED and one pushbutton in the breadboard. Wire the 5V pin to the breadboard power rail and the ground pin to the ground rail, and place the pushbutton in the center of the breadboard. Place the LED to one side of the breadboard, as shown in Figure 2-16.

Figure 2-16. *Diagram of an LED with a pushbutton*

■ **Tip** If you power on your shiny new Arduino, you may see the LED on the board flash. This is because some Arduino boards come with the blink sketch preloaded.

You're almost there. Now wire a jumper from the power rail to one side of the pushbutton, and wire the other side of the pushbutton to (DIGITAL) pin 2 on the Arduino (located on the side with the USB connector). Next, wire the LED to ground on the breadboard and a 150 ohm resistor (colors: brown, green, brown, gold). The other side of the resistor should be wired to pin 13 on the Arduino. You also need a resistor to pull the button low when the button isn't pressed. Place a 10K ohm resistor (colors: brown, black, orange, gold) on the side of the button with the wire to pin 2 and ground.

The longest side of the LED is the positive side. The positive side should be the one connected to the resistor. It doesn't matter which direction you connect the resistor; it's used to limit the current to the LED. Check the drawing again to ensure that you have a similar setup.

■ **Note**   Most Arduino boards have an LED connected to pin 13. You reuse the pin to demonstrate how to use analog output. Thus, you may see a small LED near pin 13 illuminate at the same time as the LED on the breadboard.

Writing the Sketch

The sketch you need for this project uses two I/O pins on the Arduino: one output and one input. The output pin will illuminate the LED, and the input pin will detect the pushbutton engagement. You connect positive voltage to one side of the pushbutton and the other side to the input pin. When you detect voltage on the input pin, you tell the Arduino processor to send positive voltage to the output pin. In this case, the positive side of the LED is connected to the output pin.

As you see in the drawing in Figure 2-17, the input pin is pin 2, and the output pin is pin 13. Let's use a variable to store these numbers so you don't have to worry about repeating the hard-coded numbers (and risk getting them wrong). Use the pinMode() method to set the mode of each pin (INPUT, OUTPUT). You place the variable statements before the setup() method and set the pinMode() calls in the setup() method, as follows:

```
int led = 13;    // LED on pin 13
int button = 2;  // button on pin 2

void setup() {
  pinMode(led, OUTPUT);
  pinMode(button, INPUT);
}
```

In the loop() method, you place code to detect the button press. Use the digitalRead() method to read the status of the pin (LOW or HIGH), where LOW means there is no voltage on the pin and HIGH means positive voltage is detected on the pin.

You also place in the loop() method the code to turn on the LED when the input pin state is HIGH. In this case, you use the digitalWrite() method to set the output pin to HIGH when the input pin state is HIGH and similarly set the output pin to LOW when the input pin state is LOW. The following shows the statements needed:

```
void loop() {
  int state = digitalRead(button);
  if (state == HIGH) {
    digitalWrite(led, HIGH);
  }
  else {
    digitalWrite(led, LOW);
  }
}
```

Now let's see the entire sketch, complete with the proper documentation. Listing 2-1 shows the completed sketch.

Listing 2-1. Simple Sensor Sketch

```
/*
  Simple Sensor - MySQL for the IOT

  For this sketch, we explore a simple sensor (a pushbutton) and a simple
  response to sensor input (a LED). When the sensor is activated (the
  button is pushed), the LED is illuminated.
*/

int led = 13;     // LED on pin 13
int button = 2;   // button on pin 2

// the setup routine runs once when you press reset:
void setup() {
  // initialize pin 13 as an output.
  pinMode(led, OUTPUT);
  pinMode(button, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the state of the sensor
  int state = digitalRead(button);

  // if sensor engaged (button is pressed), turn on LED
  if (state == HIGH) {
    digitalWrite(led, HIGH);
  }
  // else turn off LED
  else {
    digitalWrite(led, LOW);
  }
}
```

When you've entered the sketch as written, you're ready to compile and run it.

Compiling and Uploading

Once you have the sketch written, test the compilation using the Compile button in the upper-left corner of the IDE. Fix any compilation errors that appear in the message window. Typical errors include misspellings or case changes (the compiler is case sensitive) for variables or methods.

After you've fixed any compilation errors, click the Upload button. The IDE compiles the sketch and uploads the compiled sketch to the Arduino board. You can track the progress via the progress bar at the lower right, above the message window. When the compiled sketch is uploaded, the progress bar disappears.

Testing the Project

Once the upload is complete, what do you see on your Arduino? If you've done everything right, the answer is nothing. It's just staring back at you with that one dark LED—almost mockingly. Now, press the pushbutton. Did the LED illuminate? If so, congratulations: you're an Arduino programmer!

If the LED didn't illuminate, hold the button down for a second or two. If that doesn't work, check all of your connections to make sure you're plugged in to the correct runs on the breadboard and that your LED is properly seated with the longer leg connected to the resistor, which is connected to pin 13.

On the other hand, if the LED stays illuminated, try reorienting your pushbutton 90 degrees. You may have set the pushbutton in the wrong orientation.

Try the project a few times until the elation passes. If you're an old hand at Arduino, that may be a short period. If this is all new to you, go ahead and push that button and bask in the glory of having built your first sensor node!

WHAT ABOUT OTHER MICROCONTROLLERS?

Yes, there are other microcontrollers to choose from than the Arduino. Indeed, some predate the Arduino. If you have had experience with these, especially if you already own some of these boards and their accessories, you should consider using them because they offer many of the same benefits as the Arduino.

Some of the more popular microcontroller alternatives include the following. I include a link for more information about each.

- *Esquillo*: This is a relatively new IOT platform that features an onboard web-based development environment for developing in Squirrel featuring an interactive debugger, cloud support, and more. It also supports some Arduino shields. (See http://esquilo.io/.)

- *mbed*: This is a new IOT platform solution (currently in beta) that uses an ARM processor. (See http://mbed.com/en/.)

- *Photon (formerly Spark)*: This is an IOT development platform sporting WiFi and even an onboard web IDE. Oh, and it is Arduino compatible. (See http://particle.io.)

- *Propeller*: This is a powerful processor and wide selection of accessories available. Many older projects use this processor, which is programmed with a C-like programming language (See http://parallax.com/catalog/microcontrollers/propeller.)

- *Teensy*: This is a small microcontroller programmed with a special USB-based loader. Since it uses the same family of processors as the Arduino, it can be programmed with the Arduino IDE (See http://pjrc.com/teensy/.)

You can find many of these at popular online electronics stores such as Sparkfun (http://sparkfun.com) and Adafruit (http://adafruit.com).

Now that you've seen a number of Arduino boards, let's discuss some common and popular hardware add-ons for the Arduino.
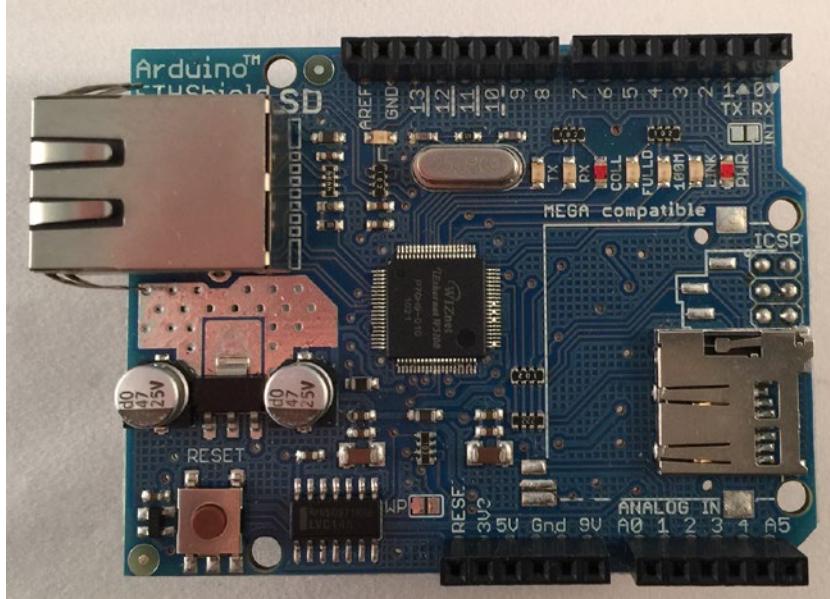
Additional Arduino Hardware

Recall the Arduino supports add-on hardware via a daughter board that fits onto the Arduino headers (called a shield). There are many such shields available ranging from simple prototyping shields to complex motor controllers. I discuss some of the shields you are likely to use in IOT solutions in this section beginning with the Ethernet shield. There are far more Arduino shields available, but I keep the discussion to these to highlight those I use most for IOT solutions.

Arduino Ethernet Shield

The Arduino Ethernet shield gives your Arduino the capability to connect to a network and ultimately the Internet via an Ethernet network connection. The Arduino Ethernet shield is made by Arduino.cc and is compatible with most Arduino boards including the large Mega.

The board also comes equipped with a MicroSD card reader, making it an excellent choice for projects that need to store or read stored data locally. There is a reset button on the board that duplicates the reset button on the Arduino. Figure 2-17 shows the Arduino Ethernet shield.

Figure 2-17. *Arduino Ethernet shield*

The board uses the built-in Ethernet library for interacting with network resources. There are a number example sketches available that you can use to learn how to use the shield as well as excellent documentation (`http://arduino.cc/en/Reference/Ethernet`).

The Arduino Ethernet shield is one of the most reproduced shields. I've found numerous third-party shields available at a fraction of the cost. Some it turns out are not 100 percent compatible, and some don't come with the MicroSD card reader. Interestingly, there is a variant of this board that permits you to use power over Ethernet (POE), allowing you to use the Ethernet cable and a special power supply on your switch or router to supply power to the Arduino.

If you need to make your Arduino solutions network aware and can use an Ethernet connection, the Arduino Ethernet shield is an excellent choice. If you want to save some money, you can choose one of the third-party versions; just make sure it is 100 percent compatible before you buy it.

A WARNING ABOUT ETHERNET SHIELD VARIANTS
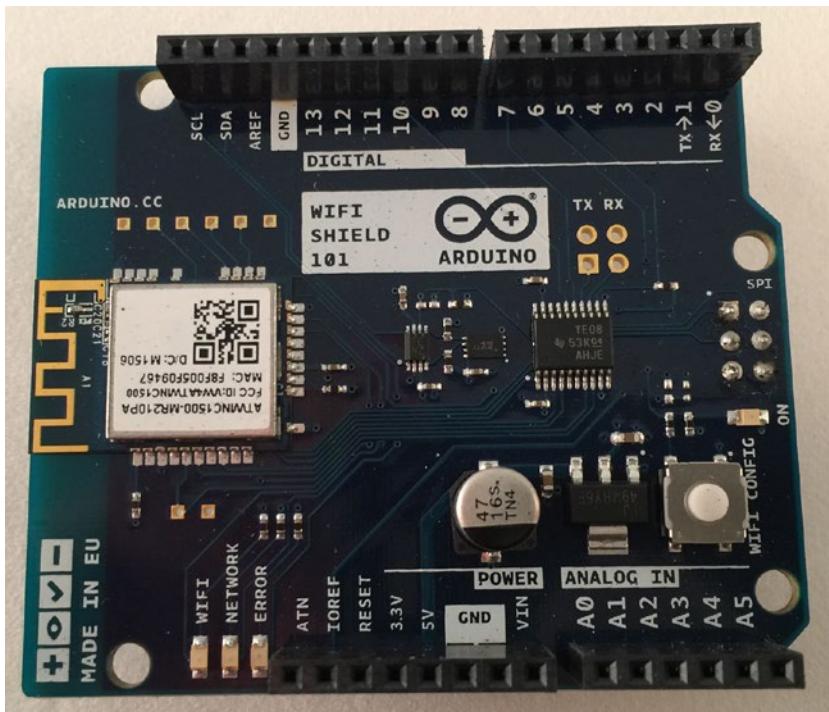
You may be thinking that all Ethernet shields are created equal and work like network cards in desktop computers. That is, any one will do; just drop it in and go. But that is far from the case. What makes an Arduino-compatible Ethernet shield is not just that it works with your Arduino; it must also use the built-in Ethernet library.

For example, there are a number of Arduino Ethernet shields and even some Ethernet modules (think breakout boards) that use a different chipset such as the CC3000 or any one of several other chipsets. Many of these are not compatible with the Ethernet library. The way you can tell is if the shield comes with or requires you to download a library to use it.

This may not be a big deal for you for a single project, but there are at least two other concerns for IOT developers. First, if you have many nodes in your solution, you could potentially have to write a different sketch for each type of Ethernet shield used. Second, if you plan to use your Arduino to talk to a MySQL server, you must have a shield that uses the Ethernet library. This is because the special library designed to talk to MySQL (called the MySQL Connector/Arduino) is written for the Ethernet library. Using a different library that does not adhere to the Ethernet.Client class may make it incompatible with the connector. You will learn more about the connector in Chapter 6.

Arduino WiFi Shield 101

The Arduino WiFi shield 101 is the latest shield from Arduino.cc for connecting an Arduino to the Internet via a WiFi signal. There are older WiFi shields available, and you will see some of those in the following sections. What makes this shield interesting and new is it has onboard crypto-authentication, making it much easier to use on wireless networks using a more secure authentication supporting both the WEP and WPA2 Security Enterprise protocols. Because of these features, it is marketed for IOT solutions. However, unlike the older Arduino WiFi shield, it does not come with a MicroSD reader. Figure 2-18 shows the Arduino WiFi shield 101.
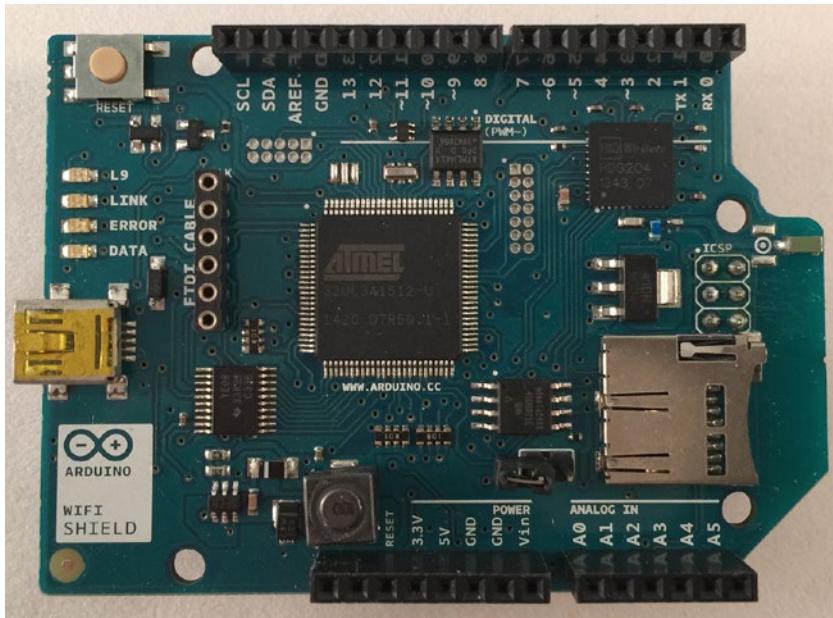
Figure 2-18. *Arduino WiFi shield 101*

There is one drawback, and it depends on how you will use the shield. Like some of the Arduino Ethernet shield clones (see the "A Warning About Ethernet Shield Variants" sidebar), it does not use the Ethernet library. You must download a new library to use it (`http://arduino.cc/en/Reference/WiFi101`). This may not be an issue if you have only a few nodes, but if you want to use the shield with existing sketches or sketches written for the Ethernet library, you may need to make some modifications to get it to work.

Fortunately, the new library has many examples you can use to write your sketches. I have not used the shield in my projects yet, but initial experimentation shows the new library is similar to the Ethernet library and thus should not be difficult to use in place of an Arduino Ethernet shield.

Arduino WiFi Shield
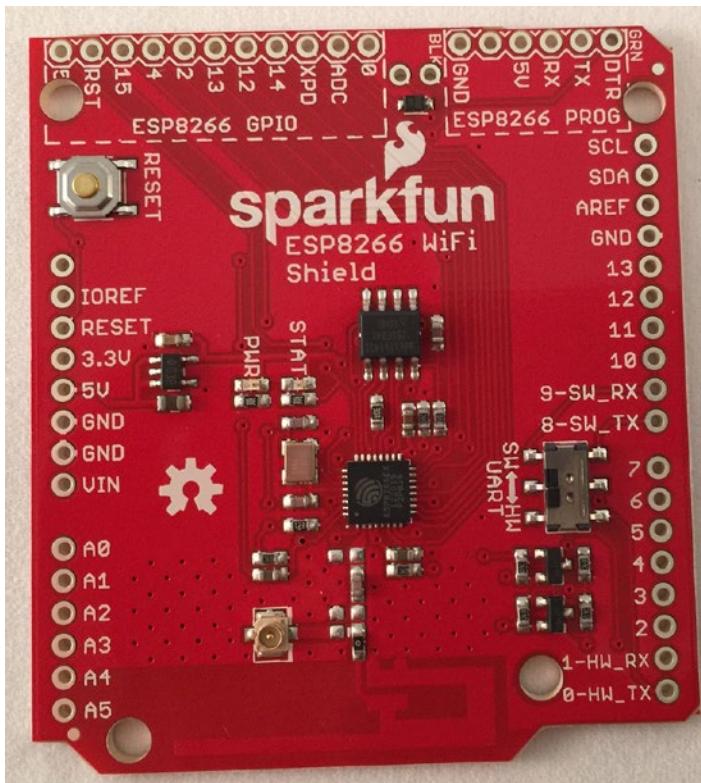
The Arduino WiFi shield is the older version of the newest 101 shield. It supports only simple authentication (no crypto features) but comes complete with a MicroSD card reader. What I like most about the shield is it is 100 percent compatible with the Ethernet library, making it easy to use the same core code in sketches for both the Arduino Ethernet and WiFi shields. There is a bit of a different startup code, but all your calls to the Ethernet library and its classes remain unchanged. Figure 2-19 shows the Arduino Ethernet shield.

Figure 2-19. *Arduino*

Sparkfun WiFi Shield: ESP8266

The Sparkfun WiFi shield, ESP8266, is an interesting board from Sparkfun (`http://sparkfun.com/products/13287`). It has the ESP8266 WiFi system on chip (SoC) processor on board, giving a split personality. Recall from the earlier section on Arduino boards, the ESP8266 WiFi SoC is a programmable module, but by placing this chip on an Arduino shield layout, it permits you to use the shield as a simple WiFi connection using the AT command set with an Arduino board. Figure 2-20 shows the Sparkfun WiFi shield, ESP8266.

Figure 2-20. *Sparkfun WiFi shield, ESP8266*

While you can use this shield as a WiFi gateway for your Arduino, it is not programmed the same way. Instead of using the Ethernet library, it has its own library that you must download and use. This library (https://github.com/sparkfun/SparkFun_ESP8266_AT_Arduino_Library) allows you to use methods to access the AT command set of the ESP8266 to connect to the network. Fortunately, the methods are similar to the Ethernet library but not compatible. Thus, you may need to rewrite the networking portions of your existing sketches to use the shield.
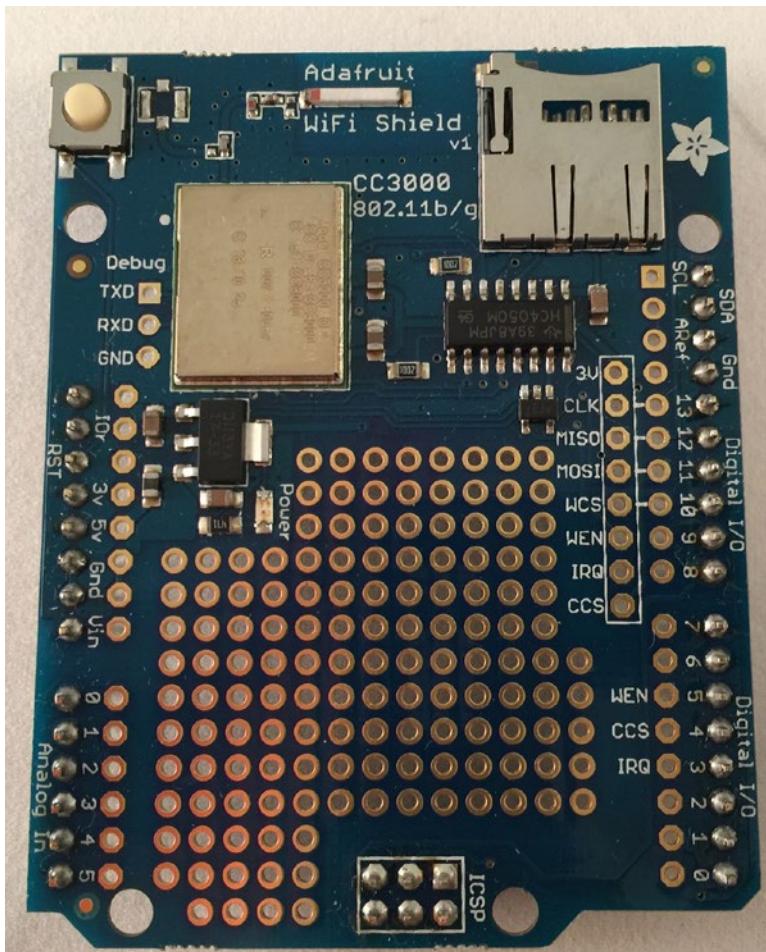
The ESP8266 WiFi SoC comes programmed to work with the Arduino as a WiFi shield. However, it can be reprogrammed. Notice the ESP8266 connections at the top of the board. You can use these to program the ESP8266. All you need to do is solder a pin header on the board and connect it via an FTDI cable. This allows you to experiment with the ESP8266 chip by modifying it to add additional commands or features, making this shield a middle ground between an Arduino and the ESP8266 breakout board.

To learn more about this special board, see https://learn.sparkfun.com/tutorials/esp8266-wifi-shield-hookup-guide.

Adafruit WiFi Shield

The Adafruit WiFi shield is one of the few Arduino Ethernet clone boards that I've found to be a good alternative. The Adafruit WiFi shield uses the CC3000 chipset, which requires a new library. Fortunately, Adafruit has an excellent tutorial for using the library (https://learn.adafruit.com/adafruit-shield-compatibility/cc3000-wifi-shield).

The shield also has a MicroSD reader, making it a good choice for me since many of my Arduino nodes use the SD card to store data for logging and as a backup for database storage. Figure 2-21 shows the Adafruit WiFi shield.
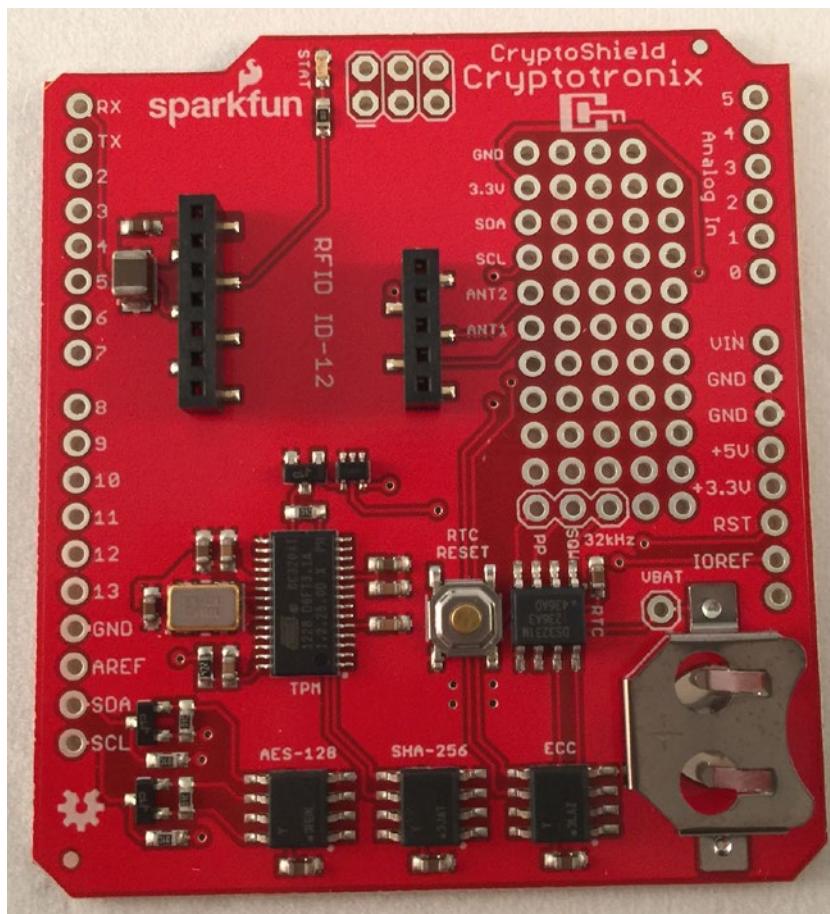
Figure 2-21. *Arduino*

Unfortunately, this shield is not compatible with the Ethernet library, which can mean rewriting the networking portion of your existing sketches. This makes it less attractive for me when developing IOT nodes because I like to use Ethernet connections when developing my sketches to eliminate the delay inherent in slower WiFi connections.

Sparkfun CryptoShield

This shield is an interesting shield. It is the first I've found that includes advanced cryptography features such as a real-time clock (RTC) module to keep accurate time, a trusted platform module (TPM) for RSA encryption/decryption, an AES-128 encrypted EEPROM for storing small amounts of encrypted data (such as user IDs and passwords for servers), support for SHA-256 and HMAC-256, and an ATECC108 that

performs the Elliptic Curve Digital Signature Algorithm (ECDSA). Wow! That's a lot of cool stuff. Figure 2-22 shows the Sparkfun CryptoShield.

Figure 2-22. *Sparkfun CryptoShield for Arduino*
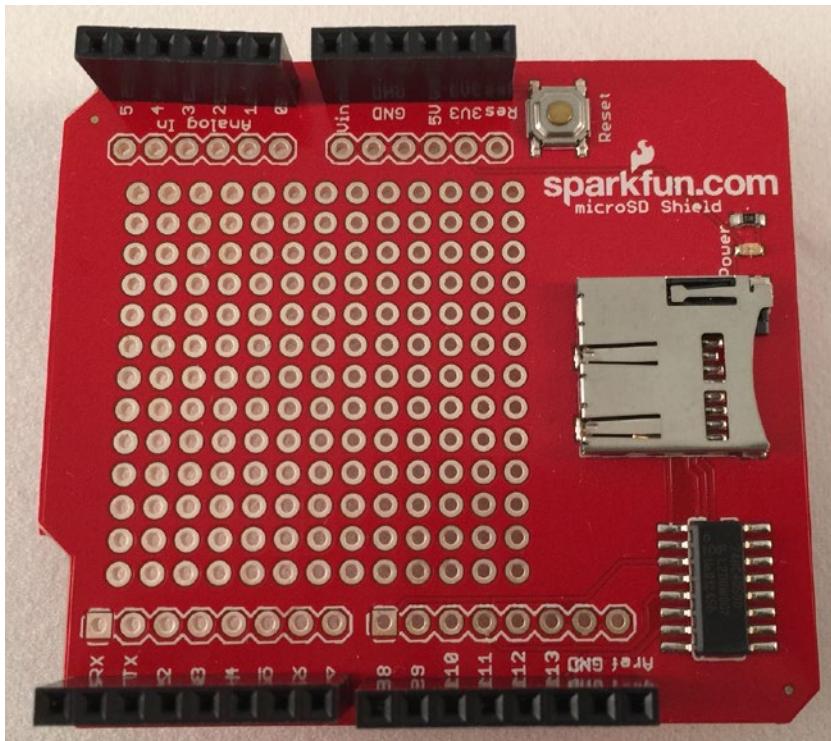
You can use this shield to encrypt data before you send it out and unencrypt it upon receipt. This may sound like overkill, but consider for the moment cases where you have nodes in your solution that must send data wirelessly or across unsecured network connections. In this case, adding hardware-level encryption means you can secure your data even if the network is not secure.

■ **Note**   Sparkfun also makes a version of this board for the BeagleBone Black.

I've used this shield with great success in securing data from one Arduino node to another. I am planning to use this shield in my IOT solutions to help secure them from surveillance or theft.

Sparkfun MicroSD Shield

The last shield in my list of example Arduino shields is the Sparkfun MicroSD shield. It is simply a shield with a MicroSD card reader. There are other variants of this shield available, but I like the Sparkfun version because it also comes with a prototyping area, making it possible to build your own circuit on the shield and giving your solution a convenient platform and a MicroSD reader to boot. Figure 2-23 shows the Sparkfun MicroSD shield.

Figure 2-23. *Arduino microSD shield from Sparkfun*

XBee Modules

The next hardware option I will discuss is not specifically an Arduino shield or even meant to be used exclusively with the Arduino. As you will see, the XBee is a versatile module used for low-cost, low-power (as in resources) communication.

The XBee is a self-contained module that uses radio frequency (RF) to exchange data from one XBee module to another. XBee modules transmit on 2.4GHz or long-range 900MHz and have their own network protocols.

The XBee module itself is small—about the size of a large postage stamp—making it easy to incorporate into small projects such as sensor nodes. The modules are also low power and can use a special sleep mode to further reduce power consumption.

Although the XBee isn't a microcontroller, it does have a limited amount of processing power that you can use to control the module. One of these features, the sleep mode, can help extend battery life for

battery-powered (or solar-powered) sensor nodes. You can also instruct the XBee module to monitor its data pins and transmit the data read to another XBee module. Aha! So, you can use XBee modules to link a sensor node to a data-aggregator node.
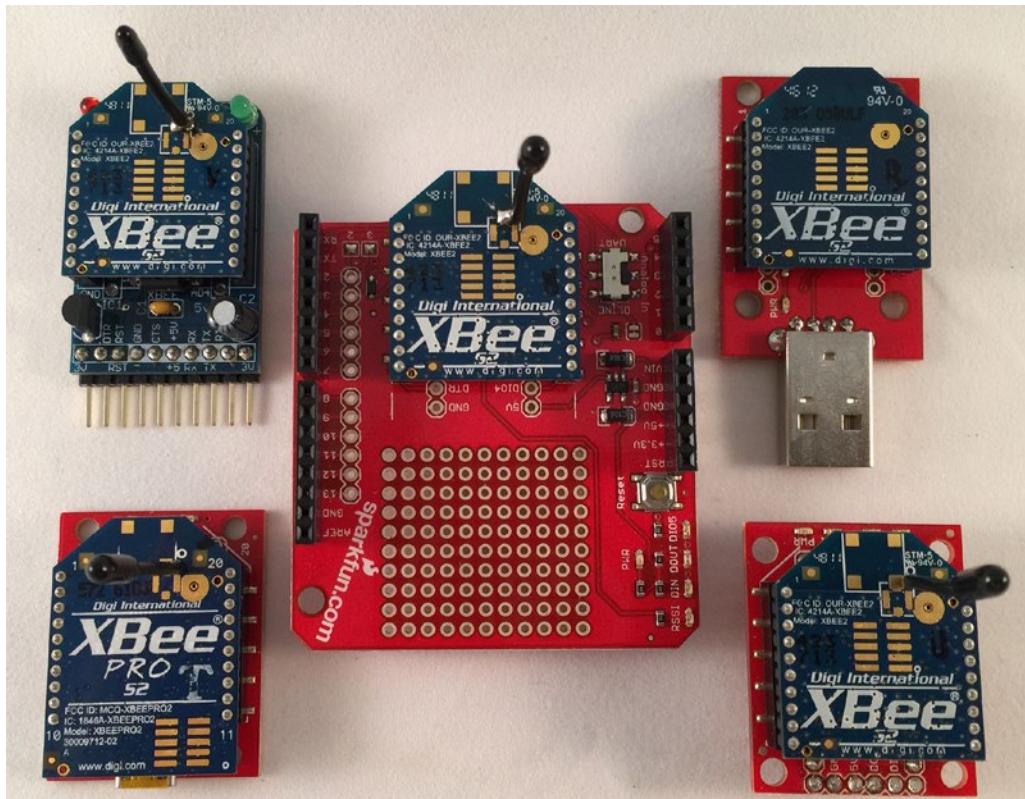
Although the XBee can be used to read sensor data, its limited processing power may mean it isn't suitable for all sensor nodes. For example, sensors that require algorithms to interpret or extrapolate meaningful data may not be suited for using an XBee alone. You may need to use a microcontroller or computer to perform the additional calculations.

■ **Tip**   I include a more complete explanation of XBee modules in my book *Beginning Sensor Networks with Arduino and Raspberry Pi* (Apress, 2013). If you want to know more about the XBee hardware and specifications, see Chapter 2 in that book.

Several XBee modules are available. There are normal models and Pro models with more power and capabilities. There is also a wide variety of antenna options from onboard chip antennas to wire whip antennas to remote antenna connections. Figure 2-24 shows several of the XBee modules I use on a regular basis.

Figure 2-24. *XBee modules and various host boards*

Notice each of the XBee modules (five in all) is mounted on a different small board. These are examples of the types of hosts you can use to program and run your XBee modules. In the center is an XBee shield from Sparkfun (`https://www.sparkfun.com/products/12847`) that allows you to connect the XBee directly to your Arduino. The rest of the boards are those you can use to program the XBee. Starting from the upper-left corner, you can see the following boards:

- *Adafruit XBee Adapter Kit*: Provides an FTDI interface to programming the XBee (`www.adafruit.com/products/126`).

- *Sparkfun XBee Explorer USB Dongle*: Allows you to plug the XBee directly into a USB port (`https://www.sparkfun.com/products/11697`).

- *Sparkfun XBee Explorer Regulated*: Provides a power-regulated breakout board (XBees run on only 3.3V, not 5V), making it possible to use this board with other discrete components such as sensors (`https://www.sparkfun.com/products/11373`).

- *Sparkfun XBee Explorer USB*: Permits you to use a USB cable with a USB cable. It also has breakout pins for the XBee, making it possible to use this board with other discrete components such as sensors (`https://www.sparkfun.com/products/11812`).

You can find XBee radios for sale at most online electronics stores such as Sparkfun and Adafruit. Aside from my sensor networks book, Adafruit (`https://learn.adafruit.com/xbee-radios`) and Sparkfun (`https://learn.sparkfun.com/tutorials/xbee-shield-hookup-guide`) have excellent tutorials on using the XBee.

Now that you've learned about the popular Arduino microcontroller, its development environment, and additional hardware (shields), let's now discover more about low-powered (also called *low-cost*) computing platforms.

Low-Powered Computing Platforms

Low-powered computing platforms, sometimes called low-cost *computer boards* or *mini-computers*,[7] are built from inexpensive components designed to run a low-resource-intensive operating system. Most boards have all the normal features you would expect from a low-cost computer including video, USB, and networking features. However, not all boards have all these features.

The reason they are sometimes called low power isn't because of their smaller CPUs or memory capabilities; rather, it is because of their power requirements, which are typically between 5V and 24V. Since they do not require a massive, PC-like power supply, these boards can be used in projects that need the capabilities of a computer with a real operating system but do not have space for a full-sized computer, cannot devote the cost of a computer, or must run on a lower voltage.

■ **Tip**   Because of the ambiguity and seeming lack of a standard nomenclature, I will refer to these boards as *low-cost computing boards* or simply *boards* when discussing them in this category.

---

[7]*Mini-computers* is not a very good name because some of these boards do not include video controllers or support for common laptop or desktop computer peripherals.

There are many varieties of low-cost computing boards. Some support the full features of a typical computer (and can be used as a pretty decent laptop alternative), while others have the bare essentials to make them usable as embedded computers. For example, some boards permit you to connect a network cable, keyboard, mouse, and monitor for use as a normal laptop or desktop computer while others have only networking and USB interfaces, requiring you to remotely access them.

One of the most powerful aspects of these boards is since they run a variant of the Linux operating system, many can be programmed with C, C++, Perl, Python, and similar programming languages. This makes it possible for you to develop your software with tools that you would find on any desktop and yet deploy it in your solution on a small, embedded computing device. Not only that, but also given the processors used are faster and some even have multiprocessing capabilities, your software will run much faster than it would on a microcontroller. How cool is that?

I have several of these boards[8] and have used them in a variety of ways. What I find most interesting about the boards I've used is some have support for Arduino shields (similar to the Arduino Yún), permitting you to mount and access an Arduino shield and connected circuitry running right on the board. I call these boards *Arduino hybrids*. I group the remaining boards into the computer boards category since they are largely full-featured computers. I discuss several examples of each category in the following sections.

Arduino Hybrids

An Arduino hybrid board is a microcontroller or a low-cost computing platform that contains Arduino headers can be used as a gateway in a sensor or IOT network where you can remotely log into it and yet access Arduino shields as if the board were an Arduino. These boards can be used in many more ways, but they excel at this role. You can set up the board on your network to give you remote access (via another computer or terminal) to the computer side while also connecting to and using the resources of the Arduino shields.

Some Arduino hybrid boards provide an onboard Arduino-compatible processor so that you can use Arduino sketches that interact with programs on the computer. These boards are a bit harder to use in the sense they don't usually support all Arduino shields, but they can be powerful when programmed using a language such as Python.

■ **Note**   Some boards, such as the Raspberry Pi, can be expanded with special add-ons (such as the Raspberry Pi AlaMode[9]) that provide similar features as the Arduino hybrids.

I present two Arduino hybrid boards: the popular pcDuino3B and the Intel Galileo.

pcDuino3B
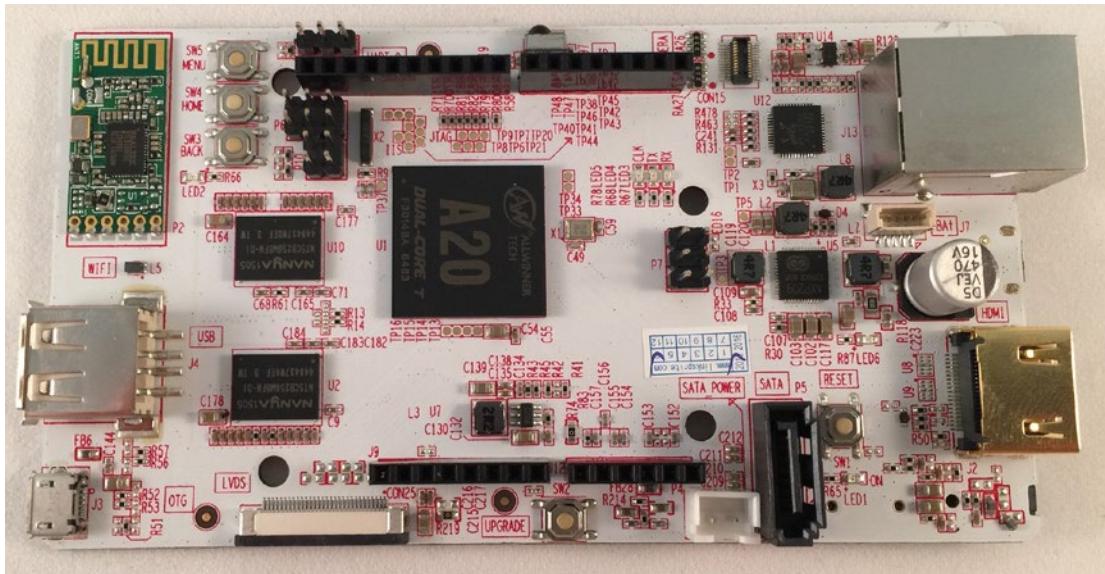
The pcDuino3B is one the more powerful boards I've used. It is effectively a full-featured small desktop computer. In fact, I've used my pcDuino3B as a laptop alternative simply by plugging in a mouse, keyboard, and monitor. Not only can you connect via WiFi to the Internet, but the board runs an older, specialized version of Ubuntu 12.04, which provides a full-featured desktop and many of the same applications you would use on a laptop or desktop. Except that there is a bare board sitting on the table, the speed of the processor makes the experience much like a laptop.

---

[8]As you will see, I have an enthusiast's passion. There's always room for one more board!
[9]http://makershed.com/products/alamode-for-raspberry-pi

The board has a powerful A20 processor (much faster than some other boards) with 1Gb of memory as well as an HDMI interface supporting an amazing 1080p resolution with its onboard video processor. It also has WiFi, Ethernet, audio, USB, a MicroSD drive, and even a SATA drive controller that permits the addition of spindle or solid-state hard drives for storage. Indeed, the feature list of this board is quite impressive.

As given the category, the board is compatible with the popular Arduino ecosystem such as Arduino shields. It also has its own header of I/O pins with 14 general-purpose I/O pins (GPIO for short), 2 PWM, 6 ADC, and 1 each of UART, SPI, and I2C pins. Figure 2-25 shows the latest pcDuino3B board.

Figure 2-25. *pcDuino3B*

■ **Note**   The pcDuino3B supports only Arduino Uno R3 3.3v shields.

I have often used the pcDuino3B as a traveling Arduino lab. Given its powerful computing features, the fact that the operating system is run from onboard memory storage and I can run the Arduino IDE from the desktop permits me to interact with the Arduino shields and hardware easily without having to lug around a separate computer, cable, Arduino, and so on. Best of all, it can be powered by USB.

I have also used the pcDuino3B as a database server (see Chapter 5) and a web server. Since it runs Ubuntu, there is little this small board cannot do. Perhaps the only downside is that it costs about twice as much as some of the other boards (but not nearly as much as the Intel boards). Thus, if you had to decide between this board and a cheaper board, it may come down to whether you need the power of the pcDuino3B versus the lower cost (and fewer features).

For more information and the full specifications of the pcDuino3B, see http://linksprite.com/wiki/index.php5?title=PcDuino3B.
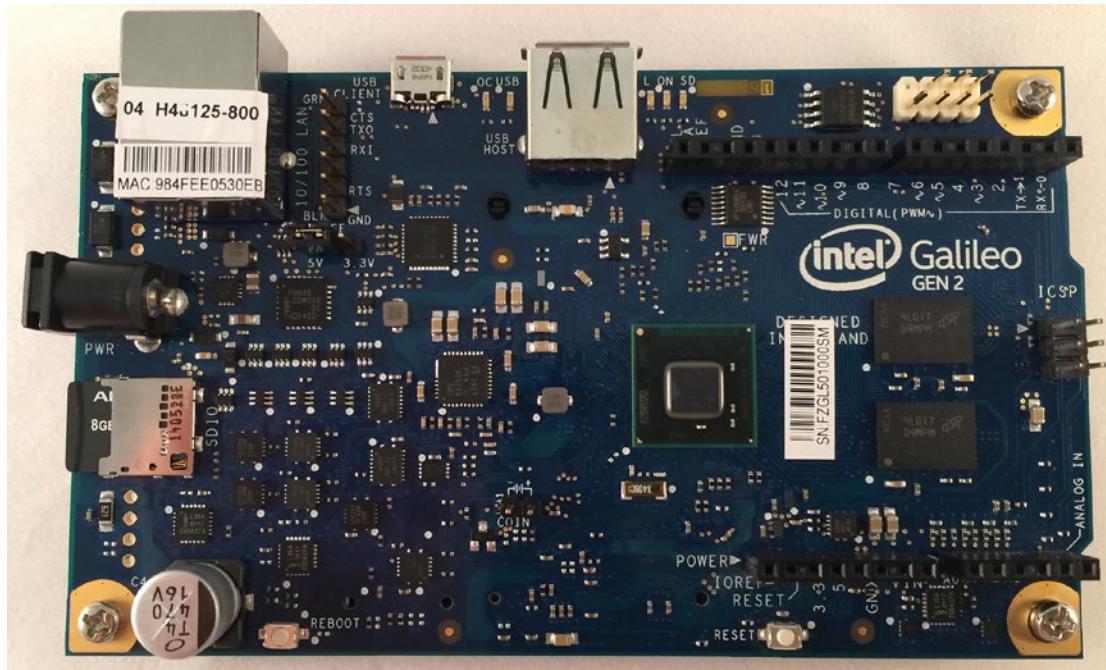
Intel Galileo Gen 2

The Intel Galileo Gen 2 is based on the Intel Quark SoC X1000 application processor, which is a 32-bit CPU based on the venerable Intel Pentium CPU packaged as a SoC. It also has Arduino headers for use with Arduino Uno R3 shields (3.3V and some 5V shields).

Perhaps its most interesting feature is that since the CPU is an Intel chip, the board can run several operating systems (but not simultaneously) such as the new Windows 10 IOT Core[10] and most Linux variants. It comes with Yocto Linux (http://yoctoproject.org/) bootable from onboard memory but can be upgraded easily.

■ **Note** Yocto Linux does not currently support as wide an array of software tools as the Ubuntu on the pcDuino. However, more packages are being added to Yocto, so chances are if you cannot find the package on Yocto you're looking for, it may be added soon.

The board has a lot of features such as USB, Ethernet, an RTC, MicroSD (used for running the operating system but unlike the pcDuino3B has a limited onboard bootable operating system), 8MB flash memory, and a mini-PCI Express slot, which promises expansion unlike any other board (but yet to be realized in the mainstream). Figure 2-26 shows the Intel Galileo Gen 2 board.

Figure 2-26. *Intel Galileo Gen 2*

---

[10]At this time, the Windows 10 IOT kit has not been officially released, but beta versions have proven interesting and promising.

The board requires external power supply and thus cannot be powered by USB but has support for power over Ethernet. While it has many interesting features, it is a relatively new addition to the market and does not seem to be catching on in popularity as much as some other boards. That said, there are several IOT expansion or experiment kits available such as the kit from SeeedStudio (http://seeedstudio.com/depot/Grove-starter-kit-plus-Intel-IoT-Edition-for-Intel-Galileo-Gen-2-and-Edison-p-1978.html).

Given the operating system is not as powerful as a full Linux implementation, you are most likely going to want to create a boot image and run a newer version of the operating system from the SD drive. You can find complete documentation on this endeavor on Intel's IOT site (https://software.intel.com/en-us/programming-blank-sd-card-with-yocto-linux-image-linux).

To date, I've used my Intel Galileo Gen 2 as a gateway for some of my Arduino shield–based circuits. This is because the board allows me to program it much like I would a normal Arduino with a special version of the Arduino IDE from Intel (http://intel.com/support/galileo/sb/CS-035101.htm). In addition, I've used the Galileo in place of an Arduino where I need more computing power or wanted easy access to the Arduino hardware from a remote computer (one I can remotely log into).

For more information about the Intel Galileo Gen 2, see https://software.intel.com/en-us/iot/hardware/galileo.

Computer Boards

The second category of low-cost computing boards are those that are intended to be embedded computing systems and do not include Arduino hardware compatibility (no shield headers or Arduino-compatible microcontroller on board).[11] However, all the boards I've used in this category support a variety of hardware pins you can use to connect sensors and other components and circuits. Thus, they are still good IOT development platforms. They just aren't Arduino hybrids.

That doesn't mean they aren't less powerful than Arduino hybrid boards (with the possible exception of the pcDuino3B); rather, they are generally more powerful and more versatile than Arduino hybrid boards. By virtue of their computer-like feel when using them, you may be surprised by their prowess.

The physical size of these boards is another characteristic that may take some getting used to. While most are very small, their size requires using both sides of the PCB for mounting components. Thus, most boards have connectors and components on both the top and the bottom. This isn't a big deal but may require some careful arranging if you are planning to mount these boards in an enclosure or may require some careful handling if you use them naked (just the bare board[12]). Fortunately, you can find custom-fit cases for most of these boards including several 3D-printable ones from http://thingiverse.com.

But they aren't just computers. Most provide GPIO pins for connecting sensors, LCDs, circuits, and more. In this respect, they are like the Arduino and can be used in many of the same situations. What makes them different is the greater processing power and connectivity that the computer aspects provide. That is, you can use a Raspberry Pi in place of an Arduino and gain the ability to remotely access that node and interact with it, which is difficult to do on the Arduino.

Boards in this category are typically computers that have dedicated hardware support for expansion, have more powerful operating system support (again, except for the pcDuino), and have networking capabilities built in. The most popular board in this category is the Raspberry Pi, which is featured predominantly later in this book. However, as you will see, there are other alternatives that are equally powerful. Let's start with a great alternative to the Raspberry Pi—the BeagleBone Black.

---
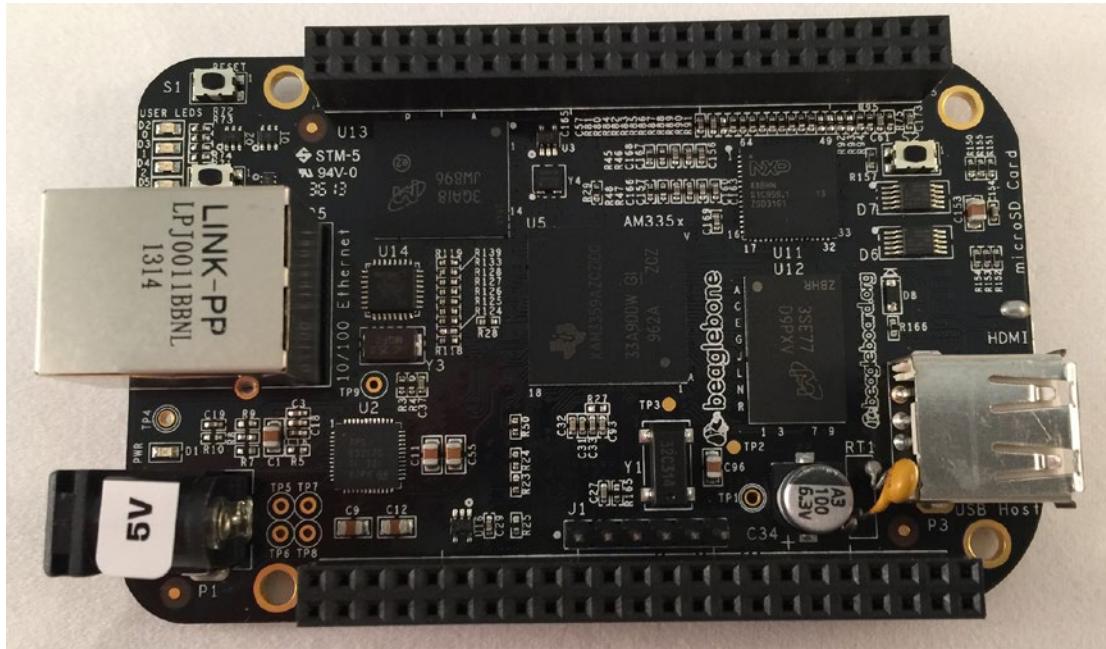
[11]The Intel Edison is a minor exception when used with the Arduino breakout host board.
[12]Cover that thing up, there may be children nearby!

BeagleBone Black

The BeagleBone Black is a small board designed for developers and hobbyists to experiment with hardware in embedded solutions using a Linux-based experience. In fact, the BeagleBone Black is designed to boot its Linux operating system in as little as ten seconds. A growing community of supporters are adding daily to the knowledge base and forum. Perhaps one of the best things about the BeagleBone Black is the startup guide at http://beagleboard.org/getting-started. It was written to get you up and running using your new "bone" in as little as five minutes. Thus, this board has a lot of appeal for someone familiar with Linux who wants to get started quickly. Figure 2-27 shows the BeagleBone Black.

Figure 2-27. *BeagleBone Black*

The board is also a capable low-cost computer with an A8 ARM processor, 512MB RAM, 4Gb of flash storage, USB host and client (for power and terminal access), a graphics chip with HDMI output, and Ethernet. There is also a power connector for sustained operation.

The footprint of the BeagleBone Black is rather unique. It can fit inside a larger metal tin like some breath mints are packaged. You would need to take a pair of tin snips to cut access for the USB, power, and Ethernet, but it will fit. In other words, it's a bit smaller than most boards.

It also has two rows of 46-pin GPIO headers for use in connecting hardware such as sensors, LCD panels, and more. In fact, you would use these headers in a similar way as you would on an Arduino or Raspberry Pi. You can also connect daughter boards called *capes* for added functionality. There are several capes available from vendors like Sparkfun, which offers the following capes:

- *4.3" LCD panel*: http://sparkfun.com/products/12085

- *7" LCD panel*: http://sparkfun.com/products/12086

- *ProtoCape for prototyping circuits*: http://sparkfun.com/products/12774

- *CryptoCape for hardware encryption*: http://sparkfun.com/products/12773
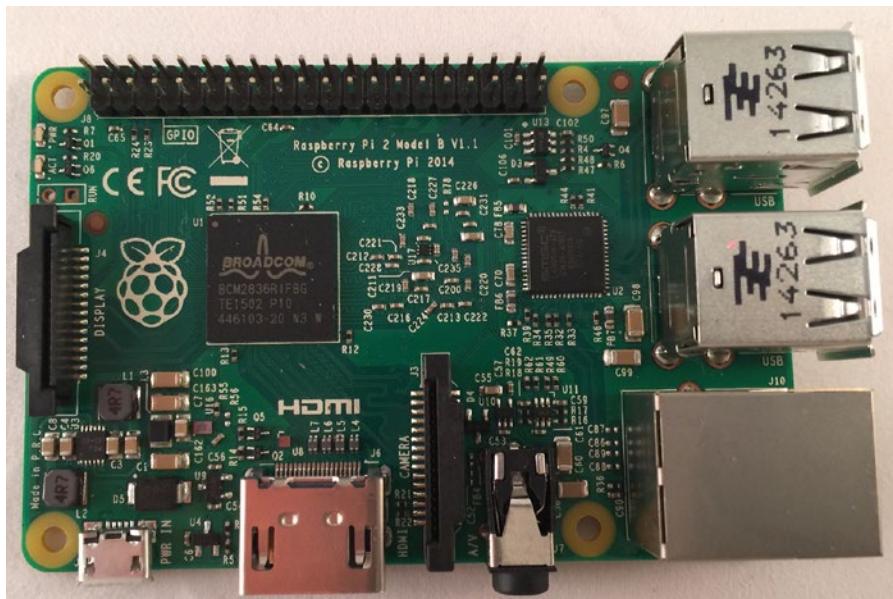
I've found the BeagleBone Black to be a capable board that can be used almost anywhere you need a more powerful processor or need to do additional processing that exceeds the capabilities of a microcontroller-based solution. I've used the BeagleBone Black in a number of experiments with hardware and find it a viable alternative to the more popular Raspberry Pi. However, in some ways the Linux feel and fit of the BeagleBone Black appeals to me more, but that is a personal choice. That said, I still have way more Raspberry Pi boards than bones.

You can learn more about the BeagleBone Black hardware including compatible hardware accessories, creating a bootable Linux image, and more at `http://elinux.org/Beagleboard:BeagleBoneBlack`.

Raspberry Pi 2 Model B

The Raspberry Pi 2 Model B is the latest iteration of the Raspberry Pi. It has all the features of the original Raspberry Pi but with a faster processor and more USB ports. The Raspberry Pi is a popular board with developers mainly because of its low cost and ease of use. Given the popularity of the Raspberry Pi, I cover it greater detail in Chapter 6, including a short tutorial on how to get started using it. Thus, I will briefly cover the highlights here and reserve a more detailed discussion on using the board for Chapter 6.

The Raspberry Pi 2B hardware includes a 900MHz A7 ARM CPU, 1GB RAM, video graphics with HDMI output, 4 USB ports (up from just 2 on older boards), Ethernet, a camera interface (CSI), a display interface (DSI), a MicroSD card, and 40 GPIO pins. Figure 2-28 shows the Raspberry Pi 2B board.

Figure 2-28. *Raspberry Pi Model 2B*

The camera interface is really interesting. You can buy a camera module like the ones at Adafruit (`http://adafruit.com/categories/177`) and connect it to the board for use as a remote video-monitoring component. I've used this feature extensively by turning a couple of my Raspberry Pi boards into 3D-printing hubs where I can send print jobs over the network to print and check the progress of the prints remotely. The software that makes this possible is called OctoPrint (`http://octoprint.org/`), and I cover it in great detail in my book *Maintaining and Troubleshooting Your 3D Printer* (Apress, 2014). See `http://apress.com/9781430268093?gtmf=s` for more details.
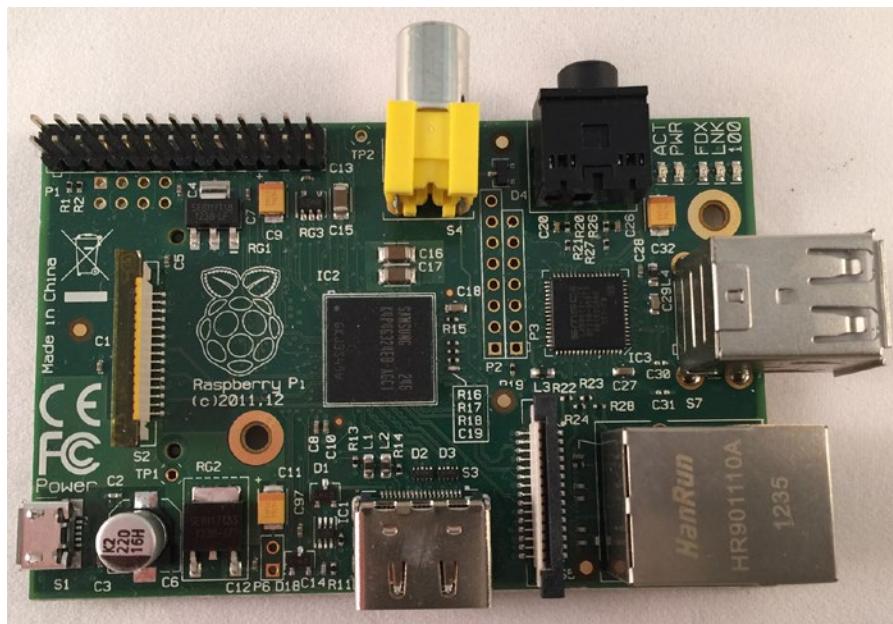
The LCD interface is also interesting because there is now a 7" LCD touch panel that connects to the DSI port (`http://element14.com/community/docs/DOC-78156/l/raspberry-pi-7-touchscreen-display`). I have yet to acquire one of these (they are in short supply), but I have some really neat ideas for a wall-mounted console for monitoring my IOT solutions. I have also seen a number of interesting Raspberry Pi tablets built using the new LCD touch panel. You can learn about one promising example (made by Adafruit, so I expect it to be excellent) at `http://thingiverse.com/thing:1082431`.[13]

Aside from that, the Raspberry Pi has been my go-to board for all manner of requirements, from a more powerful sensor node to a data aggregation node hosting a database and web server. There are also many examples from the community on how to employ the Raspberry Pi in your projects. For more information about the Raspberry Pi, see Chapter 6.

Raspberry Pi B

The Raspberry Pi B model is an older version of the Pi 2B discussed earlier with fewer features. I include it in this list because this board is plentiful and can sometimes be found at a discount compared to the newer Raspberry Pi boards. Even so, it has 512MB of RAM, two USB ports, and an Ethernet port. As a plus, accessories such as cases are also plentiful and cheaper than those for the newer models.

I have found only a few cases where I needed the more powerful Pi 2B. Thus, if you can find some of these older boards, you can save some money (that is, if you do not need the extra USB ports or other features of the P2i 2B). Figure 2-29 shows an example of an earlier version of the Raspberry Pi B.

Figure 2-29. *Raspberry Pi Model B*

---

[13]The case is a 3D-printed affair, so you'll need to find someone to print it for you if you do not have a 3D printer. A full tutorial on how to build it is included.

I often use my older Raspberry Pi B boards (I have too many it seems) whenever I need the power of the Raspberry Pi as an embedded computing node but do not need the extra features for additional USB devices, especially for cases where I can use Ethernet instead of WiFi.

I have also used this older board to prototype projects since the risk (cost) of damaging the older board is not as great as the newer boards. If I fry the older board, I just pull another one out or order another couple of used ones. If I were to damage my newest Raspberry Pi 2B, I'm out a bit more money and will likely have to wait for a new one (even today they can sometimes be hard to find).

For more information about the Raspberry Pi B, see https://www.raspberrypi.org/products/model-b/.
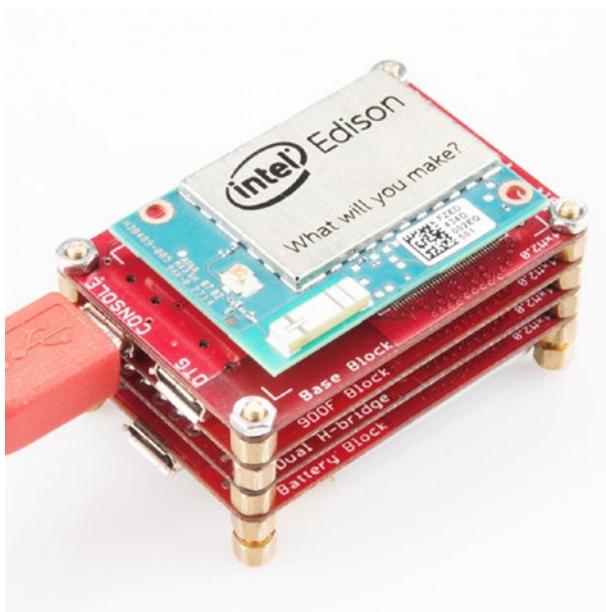
Intel Edison (with Sparkfun Blocks)

The Intel Edison is another low-cost computing board, but instead of being a tiny computer, it is more of an embedded solution than the other boards. I include it here for those who need the power of a low-cost computer in as small and as versatile a package as possible. The board is a mere 35×25×4mm in size. The Edison has an Intel Atom 500MHz dual-core, dual-threaded CPU and an Intel Quark 100MHz microcontroller. A large RFID shield hides all its components. The Edison also runs a version of the Linux operating system named Yocto, which is stored in firmware, making for very fast bootup.

Unlike the similar board, the Intel Galileo, the Intel Edison uses a single module with a micro board-to-board connector designed to allow you to stack additional boards beneath the Edison—similar to the TinyDuino. And unlike the Galileo, you can use the latest version of the Arduino IDE to program the Edison directly. You can also write programs in C, C++, or Python to run locally and access the hardware GPIO pins.

Since the Intel Edison is an embedded platform, it doesn't have USB, video, and network connections, but it does have WiFi (802.11a/b/g/n) and Bluetooth (4.0 and 2.1 EDR). You get the additional ports by using add-ons boards. After all, it is meant as an embedded IOT solution rather than a computing platform. It is also a lot smaller and therefore easier to encase into remote nodes or smaller devices.

Figure 2-30 shows the Intel Edison together with a number of add-on boards stacked beneath it. As you can see, it makes for a tidy package. Note the raised board with the Intel Edison branding. That's the Edison! Everything else in the photo is an additional component.

Figure 2-30. *Intel Edison (courtesy of http://sparkfun.com)*

Since the board comes as a SoC module, you will need to buy a host board to use it or connect it to your computer for programming or console access. Intel sells one such board, but there are several other choices from Sparkfun, as described here:

- *Intel Edison Mini Breakout*: Slightly larger than the Edison, the mini breakout provides USB ports for communication via a UART console (think terminal) and power. Intel normally sells this mini breakout board with an Edison module. See http://sparkfun.com/products/13025.

- *Intel Arduino Expansion Board*: This is a larger host board with Arduino headers for mounting Arduino shields. It provides the same USB ports as the previous host board and, like that one, is available as a kit. If you want to program the Edison with the Arduino IDE, this host board is a good choice. See http://sparkfun.com/products/13097.

- *Sparkfun Base Block*: This is a smaller host board that has the same USB ports but also comes with a pass-through micro board-to-board connector so that you can use additional boards. You can also use this board to program the Edison like an Arduino. This board is available without the Edison from Sparkfun. See http://sparkfun.com/products/13045.

- *Sparkfun Console Block*: This is similar to the Base Block but without the power port, thus providing only console access. See http://sparkfun.com/products/13039.

The modular nature of the Edison makes it possible to build your solution with only the hardware you need based on a powerful main core. This is especially true with the addition of Sparkfun's Edison Blocks. Sparkfun has created stackable modules (called *blocks*) that you can use to build a hardware stack to meet your embedded computing needs. There are quite a number of blocks available for just about whatever you want to do.

In fact, there are blocks that break out the GPIO pins in either Raspberry Pi or breadboard layouts, supply battery power, and provide a tiny OLED-based display; there's even an Arduino header block. The really cool aspect of using blocks is how they stack, which allows you to not only tailor your hardware for what you need but also to keep the hardware as small as possible to match the diminutive size of the Edison. A sample list of Sparkfun's Edison blocks is shown here:

- *Intel Edison Block ADC*: http://sparkfun.com/products/13046

- *Intel Edison Block Base*: http://sparkfun.com/products/13045

- *Intel Edison Block Dual H-Bridge*: http://sparkfun.com/products/13043

- *Intel Edison Block PWM*: http://sparkfun.com/products/13042

- *Intel Edison Block Console Basic*: http://sparkfun.com/products/13040

- *Intel Edison Block Console*: http://sparkfun.com/products/13039

- *Intel Edison Block GPIO*: http://sparkfun.com/products/13038

- *Intel Edison Block Battery*: http://sparkfun.com/products/13037

- *Intel Edison Block MicroSD*: http://sparkfun.com/products/13041

- *Intel Edison Block Arduino*: http://sparkfun.com/products/13036

- *Intel Edison Block OLED*: http://sparkfun.com/products/13035

- *Intel Edison Block I2C*: http://sparkfun.com/products/13034

- *Intel Edison Block 9 Degrees of Freedom*: http://sparkfun.com/products/13033

For more information about the Intel Edison and Sparkfun's Edison blocks, see Sparkfun's excellent guide at https://learn.sparkfun.com/tutorials/edison-getting-started-guide.

JUST HOW LOW-COST ARE THESE BOARDS?

You may be wondering about how much these boards cost. The following are the average prices for each board in the order they were mentioned. I do not include the Sparkfun Edison blocks because, as you will see, the price will depend on which blocks you buy, but generally the blocks range in price from about $15 to $35, with some kits available at a discount. I also omit the older Raspberry Pi Model B, but used prices are somewhat below the $30 mark for those boards.

- *pcDuino3B* : $60.00

- *Intel Galileo Gen 2*: $75.00

- *Raspberry Pi 2B*: $42.00

- *Intel Edison with Mini Breakout*: $75.00

- *Intel Edison with Arduino Host*: $100.00

Notice the more expensive boards are the Intel options. Given it's mostly proprietary hardware, that isn't too surprising. The real bargain is the Raspberry Pi, which partly explains its popularity. However, for the price and features, the pcDuino3B is another of my favorite boards.

Now that you've seen a variety of boards you can use to host your sensors including microcontrollers and low-cost computer boards, let's look at another key hardware component—the sensor.

Sensors

With all this talk of sensors and what sensor networks are and how they communicate data, you may be wondering what exactly sensors are and what makes them sense. This section and its subsections answer those questions and more. Let's begin with the definition of a sensor.

A *sensor* is a device that measures phenomena of the physical world. These phenomena can be things you see, such as light, smoke, water vapor, and so on. They can also be things you feel, like temperature, electricity,[14] water, wind, and so on. Humans have senses that act like sensors, allowing us to experience the world around us. However, there are some things your sensors can't see or feel, such as radiation, radio waves, voltage, and amperage. Upon measuring these phenomena, it's the sensors' job to convey a measurement in the form of either a voltage representation or a number.

There are many forms of sensors. They're typically low-cost devices designed for a single purpose and with a limited capability for processing. Most simple sensors are discrete components; even those that have more sophisticated parts can be treated as separate components. Sensors are either analog or digital and are typically designed to measure only one thing. But an increasing number of sensor modules are designed to measure a set of related phenomena.

---

[14]Shocking, isn't it?

Analog Sensors

Analog sensors are devices that generate a voltage range, typically between 0V and 5V. An analog-to-digital circuit is needed to convert the voltage to a number. Most microcontrollers have this feature built in, and the Arduino is a fine example. The Arduino has a limited set of pins that operate on analog data and incorporate analog-to-digital (A/D) conversion circuits.

But it isn't that simple (is it ever?). Analog sensors work like resistors and, when connected to microcontrollers, often require another resistor to "pull up" or "pull down" the voltage to avoid spurious changes in voltage known as *floating*. This is because voltage flowing through resistors is continuous in both time and amplitude. Thus, even when the sensor isn't generating a value or measurement, there is still a flow of voltage through the sensor that can cause spurious readings. Your projects require a clear distinction between OFF (zero voltage) and ON (positive voltage). Pull-up and pull-down resistors ensure that you have one of these two states. It's the responsibility of the A/D converter to take the voltage read from the sensor and convert it to a value that can be interpreted as data.

When sampled (when a value is read from a sensor), the voltage read must be interpreted as a value in the range specified for the given sensor. Remember that a value of, say, 2 volts from one analog sensor may not mean the same thing as 2 volts from another analog sensor. Each sensor's data sheet shows you how to interpret these values.

When you use a microcontroller like the Arduino, the A/D converters conveniently change the voltage into a value that uses 10 bits, resulting in an integer value between 0 and 1,023. For example, a sensor may measure phenomena in a range consisting of 200 points on a scale. The lowest value typically represents 0 and the highest 1,023. The Arduino in this case can be programmed to convert the value read from the A/D converter into a value on the sensor's scale.

As you can see, working with analog sensors is a lot more complicated than using the DHT-22 digital sensor from the previous section. With a little practice, you will find that most analog sensors aren't difficult to use once you understand how to attach them to a microcontroller and how to interpret their voltage on the scale in which the sensor is calibrated to work.

Digital Sensors

Digital sensors like the DHT-22 are designed to produce a string of bits using serial transmission (one bit at a time). However, some digital sensors produce data via parallel transmission (one or more bytes[15] at a time). As described previously, the bits are represented as voltage, where high voltage (say, 5 volts) or ON is 1 and low voltage (0 or even -5 volts) or OFF is 0. These sequences of ON and OFF values are called *discrete values* because the sensor is producing one or the other in pulses—it's either ON or OFF.

Digital sensors can be sampled more frequently than analog signals because they generate the data more quickly and because no additional circuitry is needed to read the values (such as A/D converters and logic or software to convert the values to a scale). As a result, digital sensors are generally more accurate and reliable than analog sensors. But the accuracy of a digital sensor is directly proportional to the number of bits it uses for sampling data.

The most common form of digital sensor is the pushbutton or switch. What, a button is a sensor? Why, yes, it's a sensor. Consider for a moment the sensor attached to a window in a home security system. It's a simple switch that is closed when the window is closed and open when the window is open. When the switch is wired into a circuit, the flow of current is constant and unbroken (measuring positive volts using a pull-up resistor) when the window is closed and the switch is closed, but the current is broken (measuring zero volts) when the window and switch is open. This is the most basic of ON and OFF sensors.

---

[15]This depends on the width of the parallel buffer. An 8-bit buffer can communicate 1 byte at a time, a 16-bit buffer can communicate 2 bytes at a time, and so on.

Most digital sensors are actually small circuits of several components designed to generate digital data. Unlike analog sensors, reading their data is easy because the values can be used directly without conversion (except to other scales or units of measure). Some may suggest this is more difficult than using analog sensors, but that depends on your point of view. An electronics enthusiast would see working with analog sensors as easier, whereas a programmer would think digital sensors are simpler to use.

So, what do you do with the data once it's measured? The following section briefly describes some aspects of sensor data and considerations for storing that data.

Storing Sensor Data

Storing sensor data depends on how the data is interpreted and ultimately how it will be used. If you plan to use a computer—or, better, a database—to store the data, you should store it in a way that makes sense.

For example, storing a sequence of voltages from an analog signal may be considered preserving the data in its purest form, but without context or an A/D converter, the data may be meaningless. Storing the digital conversion of the voltage may not be wise either, because you have to remember the scale and range in order to derive the values intended to be represented. Thus, it makes much more sense to store the resulting conversion to scale. Fortunately, when you're using digital sensors, the only thing you need to remember is what unit of measure is being used (Celsius, Fahrenheit, feet, meters, and so on). Therefore, it's best to save the final form of the measurement.

But where do you store this information? Commercial sensor networks store the data in an embedded database or file-storage device, transmit it to another system for storage, or store it on removable digital media. Older sensor networks (like a polygraph or EKG machine) store the data as hard copy using graphs (making them very obsolete).

There are a number of simple storage devices and technologies you can use to build your own sensor networks, ranging from local devices for the Arduino to modern hard drives on the Raspberry Pi. These storage mechanisms are listed here and discussed in more detail in Chapter 3.

Let's take a look at some of the sensors available and the types of phenomena they measure.

Examples of Sensors

All sensor networks begin with one sensor and a means to read and interpret the data. This chapter has presented a lot of information about sensors. You may be thinking of all manner of useful things you can measure in your home or office or even in your yard or surroundings. You may want to measure the temperature changes in your new sun room, detect when the mail carrier has tossed the latest circular in your mailbox, or perhaps keep a log of how many times your dog uses his doggy door. I hope that by now you can see these are just the beginning when it comes to imagining what you can measure. You should be thinking about what kind of sensor network you want to build; you can use this book as a means to learn how to build it.

What types of sensors are available? The following list describes some of the more popular sensors and what they measure. This is just a sampling of what is available. Perusing the catalogs of online electronics vendors such as Mouser Electronics (http://mouser.com), SparkFun Electronics (sparkfun.com), and Adafruit Industries (http://adafruit.com/) will reveal many more examples.

- *Accelerometers*: These sensors measure motion or movement of the sensor or whatever it's attached to. They're designed to sense motion (velocity, inclination, vibration, and so on) on several axes. Some include gyroscopic features. Most are digital sensors. A Wii Nunchuck (or WiiChuck) contains a sophisticated accelerometer for tracking movement. Aha: now you know the secret of those funny little thingamabobs that came with your Wii.

- *Audio sensors*: Perhaps this is obvious, but microphones are used to measure sound. Most are analog, but some of the better security and surveillance sensors have digital variants for higher compression of transmitted data.

- *Barcode readers*: These sensors are designed to read barcodes. Most often, barcode readers generate digital data representing the numeric equivalent of a barcode. Such sensors are often used in inventory-tracking systems to track equipment through a plant or during transport. They're plentiful, and many are economically priced, enabling you to incorporate them into your own projects.

- *RFID sensors*: Radio frequency identification uses a passive device (sometimes called an *RFID tag*) to communicate data using radio frequencies through electromagnetic induction. For example, an RFID tag can be a credit-card-sized plastic card, a label, or something similar that contains a special antenna, typically in the form of a coil, thin wire, or foil layer that is tuned to a specific frequency. When the tag is placed in close proximity to the reader, the reader emits a radio signal; the tag can use the electromagnet energy to transmit a nonvolatile message embedded in the antenna, in the form of radio signals that are then converted to an alphanumeric string.[16]

- *Biometric sensors*: A sensor that reads fingerprints, irises, or palm prints contains a special sensor designed to recognize patterns. Given the uniqueness inherit in patterns such as fingerprints and palm prints, they make excellent components for a secure access system. Most biometric sensors produce a block of digital data that represents the fingerprint or palm print.

- *Capacitive sensors*: A special application of capacitive sensors, pulse sensors are designed to measure your pulse rate and typically use a fingertip for the sensing site. Special devices known as pulse oximeters (called *pulse-ox* by some medical professionals) measure pulse rate with a capacitive sensor and determine the oxygen content of blood with a light sensor. If you own modern electronic devices, you may have encountered touch-sensitive buttons that use special capacitive sensors to detect touch and pressure.

- *Coin sensors*: This is one of the most unusual types of sensors.[17] These devices are like the coin slots on a typical vending machine. Like their commercial equivalent, they can be calibrated to sense when a certain size of coin is inserted. Although not as sophisticated as commercial units that can distinguish fake coins from real ones, coin sensors can be used to add a new dimension to your projects. Imagine a coin-operated WiFi station. Now, that should keep the kids from spending too much time on the Internet!

- *Current sensors*: These are designed to measure voltage and amperage. Some are designed to measure change, whereas others measure load.

- *Flex/Force sensors*: Resistance sensors measure flexes in a piece of material or the force or impact of pressure on the sensor. Flex sensors may be useful for measuring torsional effects or as a means to measure finger movements (like in a Nintendo Power Glove). Flex-sensor resistance increases when the sensor is flexed.

---

[16]http://en.wikipedia.org/wiki/Radio-frequency_identification
[17]www.sparkfun.com/products/11719

- *Gas sensors*: There are a great many types of gas sensors. Some measure potentially harmful gases such as LPG and methane and other gases such as hydrogen, oxygen, and so on. Other gas sensors are combined with light sensors to sense smoke or pollutants in the air. The next time you hear that telltale and often annoying low-battery warning beep[18] from your smoke detector, think about what that device contains. Why, it's a sensor node!

- *Light sensors*: Sensors that measure the intensity or lack of light are special types of resistors: light-dependent resistors (LDRs), sometimes called *photo resistors* or *photocells*. Thus, they're analog by nature. If you own a Mac laptop, chances are you've seen a photo resistor in action when your illuminated keyboard turns itself on in low light. Special forms of light sensors can detect other light spectrums such as infrared (as in older TV remotes).

- *Liquid-flow sensors*: These sensors resemble valves and are placed in-line in plumbing systems. They measure the flow of liquid as it passes through. Basic flow sensors use a spinning wheel and a magnet to generate a Hall effect (rapid ON/OFF sequences whose frequency equates to how much water has passed).

- *Liquid-level sensors*: A special resistive solid-state device can be used to measure the relative height of a body of water. One example generates low resistance when the water level is high and higher resistance when the level is low.

- *Location sensors*: Modern smartphones have GPS sensors for sensing location, and of course GPS devices use the GPS technology to help you navigate. Fortunately, GPS sensors are available in low-cost forms, enabling you to add location sensing to your sensor network. GPS sensors generate digital data in the form of longitude and latitude, but some can also sense altitude.

- *Magnetic-stripe readers*: These sensors read data from magnetic stripes (like that on a credit card) and return the digital form of the alphanumeric data (the actual strings).

- *Magnetometers*: These sensors measure orientation via the strength of magnetic fields. A compass is a sensor for finding magnetic north. Some magnetometers offer multiple axes to allow even finer detection of magnetic fields.

- *Proximity sensors*: Often thought of as distance sensors, proximity sensors use infrared or sound waves to detect distance or the range to/from an object. Made popular by low-cost robotics kits, the Parallax Ultrasonic Sensor uses sound waves to measure distance by sensing the amount of time between pulse sent and pulse received (the echo). For approximate distance measuring,[19] it's a simple math problem to convert the time to distance. How cool is that?

- *Radiation sensors*: Among the more serious sensors are those that detect radiation. This can also be electromagnetic radiation (there are sensors for that too), but a Geiger counter uses radiation sensors to detect harmful ionizing. In fact, it's possible to build your own Geiger counter using a sensor and an Arduino (and a few electronic components).

---

[18]I for one can never tell which detector is beeping, so I replace the batteries in all of them.

[19]Accuracy may depend on environmental variables such as elevation, temperature, and so on.

- *Speed sensors*: Similar to flow sensors, simple speed sensors like those found on many bicycles use a magnet and a reed switch to generate a Hall effect. The frequency combined with the circumference of the wheel can be used to calculate speed and, over time, distance traveled. Yes, a bicycle computer is yet another example of a simple sensor network: the speed sensor on the wheel and fork provides the data for the monitor on your handlebars.

- *Switches and pushbuttons*: These are the most basic of digital sensors used to detect if something is set (ON) or reset (OFF).

- *Tilt switches*: These sensors can detect when a device is tilted one way or another. Although simple, they can be useful for low-cost motion-detection sensors. They are digital and are essentially switches.

- *Touch sensors*: The touch-sensitive membranes formed into keypads, keyboards, pointing devices, and the like are an interesting form of sensor. You can use touch-sensitive devices like these for sensor networks that need to collect data from humans.

- *Video sensors*: As mentioned previously, it's possible to obtain small video sensors that use cameras and circuitry to capture images and transmit them as digital data.

- *Weather sensors*: Sensors for temperature, barometric pressure, rainfall, humidity, wind speed, and so on, are all classified as weather sensors. Most generate digital data and can be combined to create comprehensive environmental sensor networks. Yes, it's possible to build your own weather station from about a dozen inexpensive sensors, an Arduino (or a Raspberry Pi), and a bit of programming to interpret and combine the data.

Computer Systems

A complete discussion of IOT hardware (at least hardware for the nodes in the network) would be incomplete without mentioning computer systems. Desktop or server computers can be used in your IOT solutions in a number of ways, from providing a reliable platform for a database server, web server, firewall to the Internet, or cloud gateway (or all of these).

I typically try to avoid using computer systems in my IOT solutions mainly to keep cost down but also because low-cost computer boards and microcontroller boards are more than adequate for the solutions I've built (or dream about building).

However, if you do want to use a computer system in your IOT solution, you should do so. Just consider the added cost (even small shoebox computers are many times the cost of a Raspberry Pi), physical size and mounting or location, and additional power requirements (they're not easy to run off of batteries or solar power).

Finally, make sure you take necessary precautions in securing your computer systems so that they are not vulnerable to intrusion or invite attacks. Hackers can generally wreak much more havoc with a computer system than, say, an Arduino. However, don't dismiss low-cost computer boards as safe; they aren't. Most are also good targets for hackers given they often run powerful Linux operating systems.

Summary

The hardware available for IOT solutions includes any discrete electronic component used to create circuits for sensors, power, and many other needs. Within that vast category of electronics are the components used to support features for IOT solutions. More specifically, options are available for building a network of nodes that observe, record, and display information about the world around us.

In this chapter, you examined a long list of microcontrollers focusing on the Arduino line of microcontroller boards, low-cost (low-power) computing boards such as the Raspberry Pi, and the various communication hardware you can use to connect the nodes from using an Ethernet or WiFi network connecting to the Internet to low-cost, low-power wireless communication modules (XBee radios) for connecting sensor nodes.

You also saw a brief tutorial on how to use the Arduino programming environment and even saw an example of a simple sensor-based Arduino project (recall a switch is a simple sensor).

Finally, you took a brief look at sensors including an overview of the types and various sensors available for you to use in your IOT solution.

In the next chapter, you'll examine how the data in an IOT solution can be stored (be that on a local device such as memory-attached file-based solutions), sent to other nodes in a network, or saved to a database server.

CHAPTER 3

■ ■ ■

How IOT Data Is Stored

Ask any accomplished, professional software engineer, developer, architect, or project lead and they will tell you the key to a well-performing solution starts with a well-designed and tested plan for designing and implementing the data storage element of the project. While there are many other aspects that are equally important to quality and success, the data storage element is definitely among the top contenders for success.

The same is true for IOT solutions. This goes beyond the initial question of where to store the data. To be successful, your plan for developing the solution must also consider what will be stored and, more importantly, how it will be stored.

I've seen many hobbyist solutions that store data in a variety of ways. Some if not most of the solutions work well and have little issue related to the data—that is, until there is a problem such as needing to modify the data, recover the data, or add features to store other data. In these cases, it rapidly becomes clear the data storage component is not up to the task.

Sometimes the result is a need to change how the data is stored, making it incompatible with earlier versions of the software.[1] Other times, it requires a redesign with far more effort than should be needed or even expected, leading to delays completing the project. The root cause of these maladies is typically a poorly or under-designed data storage component.

For example, consider the implications if a solution stored data as text in a file (like a log). Sure, the data is there and you can easily write a program to read it, but the solution doesn't scale to cases where the data gets very large (many thousands or even millions of rows). Furthermore, storing the data in a file means every piece of data must be converted to a string, requiring conversion back to its original type before any mathematical operations are performed. Perhaps worse, there is no easy way to perform any sort of ad hoc query on the data. That is, to perform a query, you must augment the program code used to read the data, writing specialized code to examine the data.

In situations like this, more thought needs to be put into how the data will be used as well as how to store the data. You will examine these topics in more detail in this chapter.

■ **Note** The examples are simplistic in an effort to[2] make them easier to follow. A brief overview of the Arduino IDE and program was presented in the previous chapter.

Let's begin with a review of how IOT solutions, specifically distributed IOT solutions, are formed from several types of nodes in network architectures.

---

[1]Sound familiar? I've had this happen with far too many applications. While it is not always possible, a good data storage design should be extensible.

[2]Sometimes sacrificing efficiency or preferred techniques for easier-to-read code.

Distributed IOT

IOT solutions can be designed and assembled in many ways. Some solutions use a single component design housing all the hardware in a single box. While this design philosophy seems to be common for early IOT solutions, there is another design philosophy that enables more versatility, expandability, and features than a single hardware solution can provide.
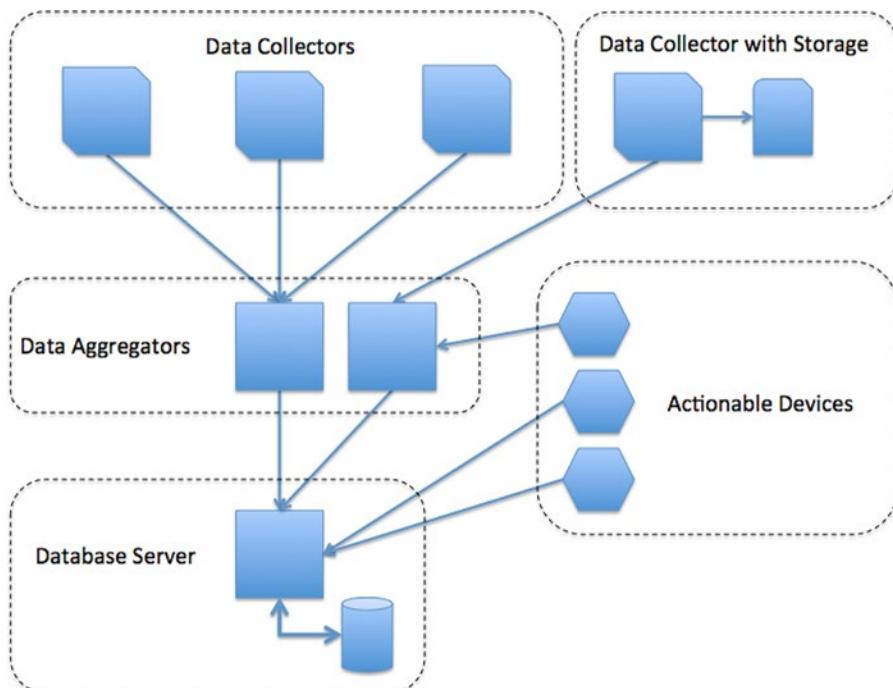
This philosophy is borrowed (somewhat) from sensor networks where the solution is composed of multiple, distributed components. In a distributed solution, the components communicate with each other using one or more network protocols. You saw this in the previous chapter where we discussed how data is collected and passed among the nodes in the network.

In this section, we discuss a distributed IOT solution. These solutions are composed of one or more data collectors with one or more sensors using a communication method or protocol to transmit the data. As mentioned, the communication method can use a device such as a microcontroller (for example, an Arduino), an embedded system, or even a small-footprint computer such as a Raspberry Pi.

Typically, the data collectors (called *sensor nodes* in sensor networks) are designed for unattended operation; they're sometimes installed on mobile objects or in locations where wired communication is impractical. In these situations, data collectors can be designed to operate without being tethered to a power (run off a battery or solar power) or communication source (using a wireless mechanism). The receivers of the data collectors can be nodes that process the data and store the data (data aggregator node) or a single database server.

While you saw an overview of each of these types of nodes in Chapter 1, this section presents more details about how each node is used to form a network to gather, transmit, augment, and store data. As you will see, I've divided some of the categories to further define the types of nodes.

Figure 3-1 shows how each type of node would be used in a fictional IOT solution. We will discuss each in more detail in the following sections.

Figure 3-1. *IOT distributed network nodes*

In this example, several data collector nodes at the top send data wirelessly to data aggregator nodes (sometimes called *data nodes*) in the middle. The data node collects the data and saves it to a secure digital card and then communicates with a database server via a wired computer network to store the data. Storing the data on the intermediate data aggregator nodes ensures that you won't lose any data if your database server fails or the network goes down.

Data Collectors

At the lowest (or leaf) level of the network is a data collector. It has at least one sensor and a communication mechanism, typically a wireless protocol. These nodes don't store or manipulate the captured data in any way—they simply pass the data to another node in the network.

Data Collectors with Storage

The next type of node is a data collector node that stores data. While these nodes may send the data to another node, typically they're devices that save the data to a storage mechanism such as a data card, to a database via a link with a desktop or server computer, or directly to a visual output device like an LCD screen, a panel meter, or LED indicators.

Data nodes require a device that can do a bit more than simply pass the data to another node. They need to be able to record or present the data. This is an excellent use for a microcontroller, as you'll see in later chapters. Data nodes can be used to form autonomous or unattended sensor networks that record data for later archiving.

For example, consider a fish or garden pond. There are many commercial pond-monitoring systems that employ self-contained sensor devices with multiple sensors that send data to a data node; the user can visit the data node and read the data for use in analysis on a computer.

Actionable Device

An actionable device is another node that is similar to a data collector and indeed may have data collection features. However, unlike data collectors that only observe and send data, actionable devices can be controlled directly or given commands to execute. For example, a camera with pan and tilt capabilities can produce a video stream or still photos of a wide area as well as receive commands for the pan and tilt features.

Typically, actionable devices require another node in the network to receive and transmit the commands. This could be a forward-facing (as in the Internet) computer or microcontroller or a remotely mounted control panel such as a tablet.

Data Aggregators

Another type of node is an aggregate node. These nodes typically employ a communication device and a recording device (or gateway) and no sensors. They're used to gather data from one or more data collectors or other data aggregator nodes. In the examples discussed thus far, the monitoring system would have one or more aggregator nodes to read the data from sensors.

Data aggregators can be used to augment the data either by adding logic that categorizes the data, adding date and time information, or even performing transformations on the data before it is saved.

Data aggregators can also be used to store a temporary copy of the data before it is sent to a database server. This allows for some minor recoverability or continued data collection should the network encounter failure in one or more nodes or even the communication protocols employed.

Database Server

The database server node is exactly what it sounds like—a computer hosting a database server that can be used to save and retrieve data from the rest of the nodes in the network. While some solutions use a dedicated database server (the networks I design use such), some solutions that support Internet-accessible features such as a web site or control panel will place the database server on the same machine as the application (for example, web server).

However, this is generally considered a potential vulnerability. That is, if the web server is compromised, the likelihood of the database server being compromised is considerable. It is best to keep the database server on a separate node if possible. This vulnerability is lessened if the solution is isolated from other networks or the Internet.

Now that you know what nodes make up a distributed IOT solution, let's look at how you can store data throughout the network. I focus on the database server option last but present a few alternatives first since, depending on the needs of the solution, some local storage may be warranted. For example, storing a copy of the actions taken on the data locally may help diagnose problems.

■ **Note** Since most data collected is sensor data, the following storage options use sensor data to illustrate the concepts.

Local On-Device Storage

Storing data on a local device such as an SD card, hard drive, electronic memory, and so on, can be complicated depending on what the data represents. For example, sensor data can come in several forms. Sensors can produce numeric data consisting of floating-point numbers or sometimes integers. Some sensors produce more complex information that is grouped together and may contain several forms of data. Knowing how to interpret the values read is often the hardest part of using a sensor. In fact, you saw this in a number of the sensor node examples. For example, the temperature sensors produced values that had to be converted to scale to be meaningful.

Although it is possible to store all the data as text, if you want to use the data in another application or consume it for use in a spreadsheet or statistical application, you may need to consider storing it either in binary form or in a text form that can be easily converted. For example, most spreadsheet applications can easily convert a text string like 123.45 to a float, but they may not be able to convert 12E236 to a float. On the other hand, if you plan to write additional code for your Arduino sketches or Raspberry Pi Python scripts to process the data, you may want to store the data in binary form to avoid having to write costly (and potentially slow) conversion routines.

But that is only part of the problem. Where you store the data is a greater concern. You want to store the data in the form you need but also in a location (on a device) that you can retrieve it from and that won't be erased when the host is rebooted. For example, storing data in main memory on an Arduino is not a good idea. Not only does it consume valuable program space, but also it is volatile and will be erased when the Arduino is powered off.

The following sections examine several options for storing data locally. I begin with examples for the Raspberry Pi and Arduino. However, similar platforms offer the same if not similar options.

Local Storage on the Raspberry Pi

The Raspberry Pi offers a number of options for local storage. You can easily create a file and store the data on the root partition or in your home directory on the SD card. This is nonvolatile and does not affect the operation of the Raspberry Pi operating system. The only drawback is that it has the potential to result in too little disk space if the data grows significantly. But the data would have to grow to nearly 2GB (for a 2GB SD card) before it would threaten the stability of the operating system (although that can happen).

It is also possible you could have a removable drive such as a USB thumb drive or even a USB hard drive attached. Once the device and drive partitions are mounted, you can read and write files on them from the Raspberry Pi. You will see this in Chapter 5 when you discover how to build a database server using a Raspberry Pi.

Because the Raspberry Pi is effectively a personal computer, it has the capability to create, read, and write files. Although it may be possible to use an EEPROM connected via the GPIO header, given the ease of programming and the convenience of using files, there is little need for another form of storage.

The Raspberry Pi can be used with a number of programming languages. One of the most popular languages is Python. Working with files in Python is easy and is native to the default libraries. This means there is nothing you need to add to use files.

The following example demonstrates the ease of working with files in Python. One thing you will notice is that it doesn't matters where the file is located—on the SD card or an attached USB drive. You only need know the path to the location (folder) where you want to store data and pass that to the open() method.

■ **Tip** The online Python documentation explains reading and writing files in detail (http://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-files).

Writing Data to Files

This example demonstrates how easy it is to use files on the Raspberry Pi with Python. I will demonstrate how to read and write files from the logged-in user's home directory. This does not require any additional hardware or software libraries. Thus, you can execute this example on any Raspberry Pi (or any Linux-compatible system). In this example, the file you will access acts like a log. That is, you always write new data to the end of the file.

You begin by creating a file to contain the Python commands. If you do not know Python, don't worry because the commands are easy to understand and for the most part are intuitive in their usage. If you have ever written a program to read files (or a script or a command/batch file), this code will look familiar.

If you haven't used a Raspberry Pi but have a computer running Mac, Linux, or Windows, you can execute this example there as well. Just remember to change the path to the file you want to read and write to something appropriate for your system.

Start by powering on and logging into your Raspberry Pi. Then open a new file with the following command (or similar) in your home directory or some place you have read and write privileges. You can use whatever editor you want. Listing 3-1 shows the code for the example.

```
nano log_file_example.py
```

■ **Tip** Name the file with a .py extension to indicate that it is a Python script. Enter the code in Listing 3-1 in the file.

Listing 3-1. Log File Example (Raspberry Pi)

```python
from __future__ import print_function
import datetime    # date and time library

# We begin by creating the file and writing some data.
log_file = open("log.txt", "a+")
for i in range(0,10):
  log_file.write("%d,%s\n" % (i, datetime.datetime.now()))
log_file.close()

# Now, we open the file and read the contents printing out
# those rows that have values in the first column > 5
log_file = open("log.txt", "r")
rows = log_file.readlines();
for row in rows:
  columns = row.split(",")
  if (int(columns[0]) > 5):
    print(">", row, end="")

log_file.close()
```

In this example, you first import the `datetime`. You use the `datetime` to capture the current date and time. Next, you open the file (notice that you are using the current directory since there is no path specified), write ten rows to the file, and then close the file.

Notice the `open()` method. It takes two parameters—the file path and name and a mode to open the file. You use `"a+"` to append to the file (a) and create the file if it does not exist (+). Other values include `r` for reading and `w` for writing. Some of these can be combined. For example, `"rw+"` creates the file if it does not exist and allows for both reading and writing data.

■ **Note** Using write mode truncates the file. For most cases in which you want to store sensor samples, you use append mode.

Next, you close the file and then reopen it for reading. This demonstrates how you can read a file and search it for data. In this case, you read all the rows and then for each row divide (using the `split()` function) it into columns. Notice in the line that writes the data the columns are separated with a comma.

In this case, you are looking for any row that has the first column greater than five. Since the file is a text file and therefore every row read is a string, you use the `int()` function to convert the first column to an integer. Once you find the row, you print it out.

It is at this point I should mention that reading the file requires an intimate knowledge of its layout (composition) with respect to the number of columns, data types, and so on. Without this knowledge, you cannot write a program to read the data reliably.

If you are following along, go ahead and run the script. To execute the file, use the following command:

```
python ./log_file_example.py
```

If you get errors, check the code and correct any syntax errors. If you encounter problems opening the file (you see I/O errors when you run the script), try checking the permissions for the folder you are using. Try running the script a number of times and then display the contents of the file. Listing 3-2 shows the complete sequence of commands for this example running three times in succession.

Listing 3-2. Log File Example Output (Raspberry Pi)

```
$ python ./log_file_example.py
> 6,2015-10-14 20:42:33.063794
> 7,2015-10-14 20:42:33.063799
> 8,2015-10-14 20:42:33.063804
> 9,2015-10-14 20:42:33.063808
$ python ./log_file_example.py
> 6,2015-10-14 20:42:33.063794
> 7,2015-10-14 20:42:33.063799
> 8,2015-10-14 20:42:33.063804
> 9,2015-10-14 20:42:33.063808
> 6,2015-10-14 20:42:38.128724
> 7,2015-10-14 20:42:38.128729
> 8,2015-10-14 20:42:38.128734
> 9,2015-10-14 20:42:38.128739
$ python ./log_file_example.py
> 6,2015-10-14 20:42:33.063794
> 7,2015-10-14 20:42:33.063799
> 8,2015-10-14 20:42:33.063804
> 9,2015-10-14 20:42:33.063808
> 6,2015-10-14 20:42:38.128724
> 7,2015-10-14 20:42:38.128729
> 8,2015-10-14 20:42:38.128734
> 9,2015-10-14 20:42:38.128739
> 6,2015-10-14 20:42:39.262215
> 7,2015-10-14 20:42:39.262220
> 8,2015-10-14 20:42:39.262225
> 9,2015-10-14 20:42:39.262230
```

Did you get similar results? If not, correct any errors and try again until you do. Notice how the first time I ran the script I got three rows, the next six, and then nine rows. This is because the first part of the script appends data to the end of the file. If you want to start over, simply delete the file created.

As you can see from this simple example, it is easy to read and write log files using Python.

Local Storage on the Arduino

Although it is true that the Arduino has no onboard storage devices,[3] there are two ways you can store data locally on the Arduino. You can store data in a special form of nonvolatile memory or on an SD card hosted via either a special SD card shield or an Ethernet shield (most Ethernet shields have a built-in SD card drive).

■ **Note**   If you are truly inventive (or perhaps unable to resist a challenge), you can use some of the communication protocols to send data to other devices. For example, you could use the serial interface to write data to a serial device.

---

[3]Except for the new Arduino Yún, which has an SD drive and USB ports for connecting external devices. The Yún and newer boards are sure to be a game changer for the Arduino world.

The following sections discuss each option in greater detail. Later sections present small projects you can use to learn how to use these devices for storing data.

Nonvolatile Memory

The most common form of nonvolatile memory available to the Arduino is electrically erasable programmable read-only memory (EEPROM—pronounced "e-e-prom" or "double-e prom"). EEPROMs are packaged as chips (integrated circuits). As the name suggests, data can be written to the chip and is readable even after a power cycle but can be erased or overwritten.

Most Arduino boards have a small EEPROM where the sketch is stored and read during power up. If you have ever wondered how the Arduino does that, now you know. You can write to the unused portion of this memory if you desire, but the amount of memory available is small (512KB). You can also use an EEPROM and wire it directly to the Arduino via the I2C protocol to overcome this limitation.

Writing to and reading from an EEPROM is supported via a special library that is included in the Arduino IDE. Because of the limited amount of memory available, storing data in the EEPROM memory is not ideal for most data nodes. You are likely to exceed the memory available if the data you are storing is large or there are many data items per sample.

You also have the issue of getting the data from the EEPROM for use in other applications. In this case, you would have to build not only a way to write the data but also a way to read the data and export it to some other medium (local or remote).

That is not to say that you should never use EEPROM to store data. Several possible reasons justify storing data in EEPROM. For example, you could use the EEPROM to temporarily store data while the node is offline. In fact, you could build your sketch to detect when the node goes offline and switch to the EEPROM at that time. This way, your Arduino-based data node can continue to record sensor data. Once the node is back online, you can write your sketch to dump the contents of the EEPROM to another node (remote storage).

A detailed example of how to work with the EEPROM on an Arduino is beyond the scope of this book. But I wanted to discuss this option since it is a viable alternative for the Arduino and for a small amount of data can be handy. However, I have described the process in detail in Chapter 5 of my book *Beginning Sensor Networks Using the Arduino and Raspberry Pi* (Apress, 2014).

SD Card

You can also store (and retrieve) data on an SD card. The Arduino IDE has a library for interacting with an SD drive. In this case, you would use the library to access the SD drive via an SD shield or an Ethernet shield.

Storing data on an SD card is done via files and is similar in concept to the previous example. You open a file and write the data to it in whatever format is best for the next phase in your data analysis. Examples in the Arduino IDE and elsewhere demonstrate how to create a web server interface for your Arduino that displays the list of files available on the SD card.

You may choose to store data to an SD card in situations where your data node is designed as a remote unit with no connectivity to other nodes, or you can use it as a backup logging device in case your data node is disconnected or your data aggregator node goes down. Because the card is removable and readable in other devices, you can read it on another device when you want to use the data.

Using an SD card means you can move the data from the sensor node to a computer simply by unplugging the card from the Arduino and plugging it in to the SD card reader in your computer. Let's see how you can read and write data to an SD card on an Arduino. Listing 3-3 shows a complete example of the log file concept. You will need an SD card formatted as a FAT partition.

STORING DATE AND TIME WITH SAMPLES

The Arduino does not have a real-time clock (RTC) on board. If you want to store your data locally, you have to either store the data with an approximate date and time stamp or use an RTC module to read an accurate date/time value. Fortunately, there are RTC modules for use with an Arduino.

Listing 3-3. Log File Example (Arduino)

```
/**
  Example Arduino SD card log file.

  This project demonstrates how to save data to a
  microSD card as a log file and read it.
*/
#include <SPI.h>
#include <SD.h>
#include <String.h>

// Pin assignment for Arduino Ethernet shield
#define SD_PIN 4
// Pin assignment for Sparkfun microSD shield
//#define SD_PIN 8
// Pin assignment for Adafruit Data Logging shield
//#define SD_PIN 10

File log_file;

void setup() {
  char c = ' ';
  char number[4];
  int i = 0;
  int value = 0;
  String text_string;

  Serial.begin(115200);
  while (!Serial); // wait for serial to load

  Serial.print("Initializing SD card...");

  if (!SD.begin(SD_PIN)) {
    Serial.println("ERROR!");
    return;
  }
  Serial.println("done.");
```

```
  // Begin writing rows to the file

  log_file = SD.open("log.txt", FILE_WRITE);
  if (log_file) {
    for (int i=0; i < 10; i++) {
      text_string = String(i);
      text_string += ", Example row: ";
      text_string += String(i+1);
      log_file.println(text_string);
    }
    log_file.close();
  } else {
    Serial.println("Cannot open file for writing.");
  }

  // Begin reading rows from the file

  log_file = SD.open("log.txt");
  if (log_file) {
    // Read one row at a time.
    while (log_file.available()) {
      text_string = String("");

      // Read first column
      i = 0;
      while ((c != ',') && (i < 4)) {
        c = log_file.read();
        text_string += c;
        if (c != ',') {
          number[i] = c;
        }
        i++;
      }
      number[i] = '\0';
      value = atoi(number);

      // Read second column
      c = ' ';
      while (c != '\n') {
        c = log_file.read();
        text_string += c;
      }
      // If value > 5, print the row
      if (value > 5) {
        Serial.print("> ");
        Serial.print(text_string);
      }
    }
```

```
    // close the file:
    log_file.close();
  } else {
    // if the file didn't open, print an error:
    Serial.println("Cannot open file for reading");
  }
}

void loop() {
  // do nothing
}
```

■ **Note**   I omitted the writing of the date and time. There is a way to do this, but it is quite a bit more code, and I wanted to concentrate on the file operations. To see how to use the `DateTime` library on the Arduino, see http://playground.arduino.cc/Code/DateTime.

Notice there is a lot more code here than the Raspberry Pi example. This is because the Arduino code libraries do not have the same high-level primitives as Python. Thus, we have to do a lot of lower-level operations ourselves.

To keep it simple, I put the code in the setup() method so that it runs only once. Recall code in the loop() method runs repeatedly until the Arduino is powered off (or you initiate low-level code to halt execution or reboot). If you were using this example in an IOT solution, the code for initiating the SD card would remain in the setup() method and perhaps even the code to open the file, but the code to write the file would be moved to the loop() method to record data read from sensors or other data collectors.

The code example begins with some variables used for selecting the pin used for communicating with the SD card. I've included several popular options. Check the documentation for your hardware to ensure the correct pin is specified.

Next, you will see code for opening the file and writing several rows. In this case, I simply write a simple text string using the value of the counter followed by a bit of short text. As I mentioned, I did not record date and time information in this example because the Arduino does not include an RTC (although newer boards may include an RTC). Notice I open the file, write the rows, and then close the file.

Next is a block of code to read rows from the file. Like the Raspberry Pi example, we have to read the row and split the columns ourselves. However, in this case, we must read a character at a time. Thus, I use a loop to do so and terminate when the comma is located. The second part of the loop reads the remaining text from the row until the newline character is found.

I then check the value of the first column (after converting it to an integer), and if > 5, I print out the row to the serial monitor. Let's see this code in action. Listing 3-4 shows the output of the code as seen in the serial monitor.

Listing 3-4. Log File Example Output (Arduino)

```
Initializing SD card...done.
> 6, Example row: 7
> 7, Example row: 8
> 8, Example row: 9
> 9, Example row: 10
Initializing SD card...done.
> 6, Example row: 7
> 7, Example row: 8
```

```
> 8, Example row: 9
> 9, Example row: 10
> 6, Example row: 7
> 7, Example row: 8
> 8, Example row: 9
> 9, Example row: 10
Initializing SD card...done.
> 6, Example row: 7
> 7, Example row: 8
> 8, Example row: 9
> 9, Example row: 10
> 6, Example row: 7
> 7, Example row: 8
> 8, Example row: 9
> 9, Example row: 10
> 6, Example row: 7
> 7, Example row: 8
> 8, Example row: 9
> 9, Example row: 10
```

As you can see, the output is similar to the Raspberry Pi example. Clearly, either can be used to store data locally in files, but the Arduino takes a bit more work. Fortunately, the Arduino IDE includes a well-designed SD card library.

In fact, there are a number of other functions in the SD card library for dealing with files. For example, you can list the files on the card, create folders, and even truncate or delete the contents of a file. You may find this code useful in working with the previous code. The following shows how easy it is to truncate a file. If you want to use this, place it in the code before the file is opened the first time.

```
if (SD.remove("log.txt")) {
  Serial.println("file removed");
}
```

■ **Tip** For more information about using SD cards on the Arduino, see the Arduino online reference guide (http://arduino.cc/en/Reference/SDCardNotes).

Now that you've seen some of the options for storing data locally on the data collector nodes, the following section discusses sending data to data aggregators for either local or remote storage. Remote storage in this case is typically a database server.

Passing the Buck to Aggregators

Recall that a data aggregator is a special node designed to receive information from multiple sources (data collectors or sensors) and store the results either locally or remotely. The source data can originate from multiple sensors on the node itself, but more often the data aggregator receives information from multiple data collector nodes that are not attached directly to the aggregate node (they may be connected via a low-power, low-overhead communication protocol such as provided by XBee modules).

WHAT'S AN XBEE?

XBee is a brand name of Digi International for small, self-contained, modular, cost-effective components that use radio frequency (RF) to exchange data from one XBee module to another. XBee modules transmit on 2.4GHz or long-range 900MHz and have their own network protocols (ZigBee).

Although the XBee isn't a microcontroller, it does have a limited amount of processing power that you can use to control the module. One of these features, the sleep mode, can help extend battery life for battery-powered (or solar-powered) sensor nodes. You can also instruct the XBee module to monitor its data pins and transmit the data read to another XBee module. Thus, you can use an XBee module to send data from one or more sensors to a data aggregator node.

In some IOT solutions, sensors are hosted by other nodes and placed in remote locations. The data aggregator node is connected to the data collector nodes via a wired or wireless connection. For example, you may have a sensor hosted on a low-power Arduino in one location and another sensor hosted on a Raspberry Pi in another location, both connected to your data aggregator using XBee modules. Except for the limitations of the network medium chosen, you can have dozens of nodes feeding sensor data to one or more data aggregator nodes.

The use of data aggregator nodes has several advantages. If you are using a wireless technology such as ZigBee with XBee modules, data aggregator nodes can permit you to extend the range of the network by placing the data aggregator nodes nearest the sensors. The data aggregator nodes can then transmit the data to another node such as a database server via a more reliable medium.

For example, you may want to place a data aggregator node in an outbuilding that has power and an Ethernet connection to collect data from remote data collector nodes located in various other buildings.

■ **Note** In this case, I mean the closest point to the sensor nodes that is still within range of the wireless transmission media (such as ZigBee used on XBee modules).

Data aggregator nodes can also permit you to move the logic to process a set of sensors to a more powerful node. For example, if you use sensors that require code to process the values, you can use a data aggregator node to receive the raw data from those sensors, store it, and calculate the values at a later time.

Not only does this ensure that you have code in only one location, but it also allows you to use less sophisticated (less powerful) hosts for the remote sensors. That is, you could use less expensive or older Arduino boards for the sensors and a more powerful Arduino for the data aggregator node. This has the added advantage that if a remote sensor is destroyed, it is not costly to replace.

Recall also that you have to decide where you want to store your sensor data. Data aggregator nodes either can store the data locally on removable media or an onboard storage device (local storage) or can transmit the data to another node for storage (remote storage). The choice of which to use is often based on how the data will be consumed or viewed.

For example, if you want to store only the last values read from the sensors, you may want to consider some form of visual display or remote-access mechanism. In this case, it may be more cost effective and less complicated to use local storage, storing only the latest values.

On the other hand, if you require data values recorded over time for later processing, you should consider storing the data on another node so that the data can be accessed without affecting the sensor network. That is, you can store the data on a more robust system (say, a personal computer, server, or cloud-based service) and further reduce the risk of losing data should the aggregate node fail.

The nature of the local storage is a limiting factor in what you can do with a local-storage data aggregator node. That is, if you want to process the data at a later time, you would choose a medium that permits you to retrieve the data and move it to another computer.

This does not mean the local-storage data aggregator is a useless concept. Let's consider the case where you want to monitor temperature in several outbuildings. You are not using the data for any analysis but merely want to be able to read the values when it is convenient (or required).

One possible solution is to design the local-storage data aggregator node with a visual display. For example, you can use an LCD to display the sensor data. Of course, this means the data aggregator node must be in a location where you can get to it easily.

But let's consider the case where your data aggregator node is also in a remote location. Perhaps it too is in another outbuilding, but you spend the majority of your time in a different location. In this case, a remote-access solution would be best.

The design of such a data aggregator node would require storing the latest values locally, say in memory or EEPROM and, when a client connects, displaying the data. This is a simple and elegant solution for a local-storage data aggregator node.

However, it is more robust to pass the data from the data collectors to a database server. As you will see in Chapter 5, you can use a Raspberry Pi or similar low-cost board to build a database server and deploy it in your IOT solution.

The data aggregator nodes would then require a library called a *connector* to allow you to write programs (scripts, sketches) to connect to the database server and send the data. In this book, I feature the MySQL database server. In the case of the Raspberry Pi, the database connector I will demonstrate is called Connector/Python and is available from Oracle (`http://dev.mysql.com/downloads/connector/python/`). I will also discuss a database connector for the Arduino, called Connector/Arduino, which I created[4] and is available on GitHub (`https://github.com/ChuckBell/MySQL_Connector_Arduino`) or via the Library Manager in the Arduino IDE. The connector therefore forms a communication pathway to the MySQL server.

Even with the use of the connector, there are many things that you can do with a database server in your solution. Not only does a database server provide a robust storage mechanism, it can also be used to offload some of the data-processing steps from data aggregators. I discuss some of the ramifications and things to consider when using a database server node in your IOT solution.

Database Storage

A database option represents a more stable, more easily scaled, and more easily queried storage option. Indeed, that is what this book is all about—discovering how best to employ a database server in your IOT solution!

In this section, we explore some of the benefits, techniques, and considerations for using a database server in your IOT solution. As you will see, there is a lot of power available to you when you use a database server. I will give a high-level overview of the topics here with a more in-depth, hands-on explanation in Chapter 5. Let's begin by discussing why you would use a database server in your IOT solution.

■ **Tip** While an in-depth, full discussion of database design is beyond the scope of this book, the following text views the subject from a slightly different angle: how you can best design your databases for easy storage and retrieval. Thus, I assume no prior knowledge of database design. If you have database design experience, you may want to skim this section.

---

[4]Officially owned by Oracle but supported by me exclusively.

Database servers provide a structured manner in which to store and retrieve data. Interaction with a database server requires the use of a special set of commands or, more precisely, a special language that expresses storage and retrieval. The language is called Structured Query Language (SQL).[5] Most database servers use a form of SQL for most of their commands. While there are some differences from one database server to another, the syntax and indeed the core SQL commands are similar.

In this book, we use the MySQL database system as the database server. MySQL[6] is the most popular choice for developers because it offers large database system features in a lightweight form that can run on just about any consumer computer hardware. MySQL is also easy to use, and its popularity has given rise to many online and printed resources for learning and using the system.

Benefits

As mentioned, using a database server in your IOT solution has many advantages. Not only does a database server permit structured, robust storage of your data, but it also provides a powerful mechanism for retrieving data.

Consider a solution where you store data in files. As you saw in the previous sections, reading a file and looking for information requires parsing (separating the data elements) the data and then comparing the data to fit the criteria needed. The problem is each time you want to do a search you need to modify your program or script.

This doesn't sound too bad, but consider the possibility that you may need to execute searches (queries) at any time. Furthermore, consider it possible you want to be able to do this without rewriting your code or perhaps you want to allow your users to execute the queries. Clearly, using files or similar storage mechanisms does not easily permit this behavior.

This is one of the most beneficial aspects of using a database server. You can execute a query at any time (ad hoc) and you do not need a special program or need to rewrite anything to use it. I should note that some solutions hard-code their queries in their code, and some would argue that it is the same thing as accessing files. But it isn't.

In the case of a file-based solution, even if you use programming primitives, you still must write code to execute the query, whereas in a database solution, you need only to replace the query statement itself. For example, consider the Raspberry Pi file example in Listing 3-1. The code needed to execute the query (choose only those rows where the value of the first column is greater than five) is several lines long[7] and requires not only choosing the rows that match but also having to read the columns one character at a time.

Now consider the following SQL statement. Don't worry about the details of the command. Consider only that we can express the query as a single statement as follows:

```
SELECT * FROM db1.table1 WHERE col1 > 5;
```

---

[5]https://en.wikipedia.org/wiki/SQL
[6]http://dev.mysql.com/
[7]The Arduino example is three times the number of lines of code!

Notice the criteria have been moved to the database server. That is, the code doesn't need to read, interpret, and test the values for the column. Rather, that logic is executed on the database server. In fact, the database server is optimized to execute such a query in the most efficient way possible—something that would require considerable work on a file-based solution (but not unheard of). In case you are curious, the code to execute and retrieve the rows is as follows:

```
cur.execute("SELECT * FROM db1.table1 WHERE col1 > 5")
rows = cur.fetchall()
for row in rows:
    print ">", row
```

While this is a trivial example, the point is still valid. Specifically, the database server is a powerful searching tool. Not only does this allow you to simplify your code, it allows much greater expressiveness when constructing queries.

For example, you can construct queries that perform complex mathematical comparisons, text comparisons (even wildcard matching), and date comparison. For instance, you can use date operations to select rows older than N days from time of execution or rows recorded during a specific year, month, day, or hour. Clearly, this is far more powerful than code you write yourself!

Another benefit of using a database server is you can group your data in a logical manner. A database server permits you to create any number of databases for storing data. Typically, you want to create a separate database for each of your IOT solutions. This makes working with the data at a logical level easier so that data for one solution isn't intermixed with data from another. Thus, a single database server can support many IOT solutions.

Techniques

Using a database server in your solution requires using some different techniques than other storage solutions. You have already seen that file-based systems are susceptible to file layout changes (if the file layout changes, so too must the code); the same is not true for databases.

For example, if you need to add a column to the table, you don't have to rewrite code to read the data. Indeed, many changes can be made to the database without affecting the code. While some changes may affect the SQL statements, you have a great deal of freedom in how the data is stored than file-based solutions.

Thus, the biggest change in development is how you work with the database server versus the code itself. That is, code development can proceed somewhat independently of the database development. You can develop the database, its components, and the SQL statements separately from the code. Indeed, you can create a working and tested database component before you develop any of the other nodes!

This may seem like it is more work than the code for file-based solutions, but it really isn't for two reasons. First, you can test your SQL statements in isolation with test data. This means you do not need to have your entire IOT solution up and running, which makes it easier to develop. Second, and related, is you can execute your SQL statements repeatedly to ensure you get the correct data, which makes it easier to separate the data from the code and thereby makes the solution easier to maintain. That is, if something changes, you can simply change the SQL statements rather than rewrite the code.

But this does not mean you do not need to spend time designing and testing the database design. On the contrary, to reap these benefits, your database should be designed well. Database design can be complex if the data itself or your use of the data is complex. Fortunately, for most IOT solutions, this isn't a problem.

Finally, working with a database server allows you to quickly set up test data, manipulate it, and refresh it. Again, this is possible with other storage solutions, but it is so much easier with database servers. Thus, the technique for setting up test data permits you to ensure your queries return precisely what you expect. This is because you know the input (the sample data) and can easily and manually determine what the results should be.

After all, using a database server over a file-based or memory-based storage solution changes how you develop your solution by making it easier to work with the data through testing the SQL statements outside of the applications or deploying and executing the network nodes.

Considerations

Since you are reading this book, you are most likely convinced or nearly convinced you want to employ a database server in your IOT solution. But perhaps you're wondering what the ramifications or limitations are using a database server in your solution.

Perhaps the most important consideration is you need a platform on which to host the database server. If your solution employs a computer, you have everything you need. MySQL runs on commodity hardware, and for small solutions like an IOT solution, even the most basic computer is more than adequate.

However, what if you do not have a computer? In this case, you need to add hardware to host the database server. Fortunately, MySQL runs on most low-cost computer boards such as the Raspberry Pi, Beaglebone Black, pcDuino, and even the newest Intel IOT boards. While this means another node in your network, you have seen that this node fits very well into the plan (a database node). I give a complete tutorial on building a MySQL database node in Chapter 5.

Aside from adding a new node, other considerations include using SQL in your code. For this, we use a connector to connect to the MySQL server via an Ethernet connection, send queries to the server for execution, and then retrieve and process the results. If you do not know SQL, you will have to learn how to form queries. However, as you will see in Chapters 5 and 6, SQL statements are not difficult to learn and use.

Another consideration is storing data in a database server requires a good design confined to the database server terminology and features. More specifically, you have to form the layout of the tables to include selecting the correct data type from a long list of types available.

Part of this process (called *database design*) concerns designing the queries themselves or, more specifically, to design the queries so that they return exactly what you want. Thus, queries should be tested on the database server (or through a client connection, as you will see in Chapter 5) to ensure your SQL statements are correct and that there are no surprises should unusual data appear. As you saw in the previous section, the advantage is easier development of the layer of the solution that deals with the data.

When you design your tables, you should keep a few things in mind. First, consider what data types are needed for storing your samples. You should consider not only how many values each sample contains but also their format (data type). The basic data types available include integer, float, double, character, and Boolean. There are many others, including several for dates and times, as well as binary large objects (blobs) for storing large blocks of data (like images), large texts (the same as blobs, but not interpreted as binary), and much more.

You can also consider adding columns such as a timestamp field, the address of the data collector node, perhaps a reference voltage, and so on. Write all of these down, and consider the data type for each.

■ **Tip** See the online MySQL Reference Manual for a complete list and discussion of all data types (http://dev.mysql.com/doc/refman/5.7/en/).

What may be a consideration for the database design is the physical storage required for the database. If your IOT solution uses nodes that do not have any physical storage, you should add one. Not only does physical storage mean the data is not susceptible to node failure (say if memory gets erased), but it also permits you to use media with large storage capacities. While you can use a secure digital memory card, it is best to use a solid-state or older spindle disk.

Maintenance of the data is another consideration, but perhaps less so than the others mentioned. More specifically, even file-based solutions require maintenance or at least facilities for conducting maintenance such as backup and recovery. For file-based solutions, this can be simply a matter of copying files. For a database solution, backup and recovery are a bit more complicated.

Fortunately, you can use utilities such as mysqlpump[8] (or the older mysqldump[9]) or MySQL Utilities[10] to make logical backups of the data. That is, these utilities produce a file of SQL commands you can replay to create and store the data. For physical backups (at the byte level), you would have to use a commercial application such as MySQL Enterprise Backup to back up and restore the data.

Finally, security can be a consideration if your database server node is visible from outside your network. In this case, you must take care to ensure not only is the platform access (for example, the Raspberry Pi) secured but also all database security is properly designed so that users who have access to the database server have only enough permission to execute the queries for the solution and nothing more. In other words, you need to design your solution with security in mind.

Now that you've explored the database node, let's look at some best practices for designing and implementing a distributed IOT network.

Distributed IOT Network Best Practices

If your IOT solution needs to be deployed over a disperse area where the data collectors are physically separated or you want to use commodity, low-cost hardware to develop your solution, you may need to design your IOT solution using a distributed network of nodes.

This section examines some important considerations for planning the network. I discuss placement of the nodes in the network as well as design considerations for data storage. Most of these best practices are data-centric for good reason—the solution is useless without data that is accessible.

Node Placement

When planning your solution, you should consider what data you want to collect. Moreover, you should consider where and how you would collect the data. This includes where the sensors need to be located as well as what data is produced.

Placing the nodes with sensors (data collectors) is likely to be a simple choice—they need to be near the things they are observing. If the data collectors are outside, they may need weatherproof enclosures. Even inside you may need to secure the hardware to avoid accidental tampering (such as small fingers or nosy, curious friends[11]). In fact, I recommend using an appropriate enclosure for each node.

However, where you place the data aggregators may be more problematic. This is normally dictated by the communication mechanism chosen. Recall if you use a low-overhead mechanism such as ZigBee or Bluetooth, you may be limited to a certain range. Thus, the data aggregator needs to be close enough to communicate with the data collectors.

---

[8]mysqlpump is available in server versions 5.7.8 and newer (http://dev.mysql.com/doc/refman/5.7/en/mysqlpump.html).

[9]mysqldump is available in server versions 5.6 and prior (http://dev.mysql.com/doc/refman/5.7/en/mysqldump.html).

[10]Using mysqldbexport and mysqldbimport (http://dev.mysql.com/doc/mysql-utilities/1.6/en/mysqldbexport.html).

[11]I've had more experiments wrecked by a casual, "Hey, what's this?" inquiry often involving mishandling or relocation of the device.

Furthermore, the data aggregator nodes are best placed where they can communicate to the database server or visualization application (perhaps another node hosting a web server or cloud gateway) via a WiFi or Ethernet network. Thus, depending on how many data collectors you have and their physical proximity to a location where a data aggregator could be placed, you may need to employ multiple data aggregators. If the data collectors are all placed within the range of your low-cost communication mechanism, you may be able to use a single data aggregator.

Perhaps on a lesser scale are the capabilities of the data aggregator hardware. If you choose a platform that can support a limited number of connections, you are therefore limited to how many data collectors the data aggregator can support. Once again, you may need more than one data aggregator to support all the data collectors.

Another node type to consider for node placement is an actionable device. If the device produces data, you may need to connect it to a data aggregator. However, if the device has the ability to be programmed or to run scripts, you may be able to program it to write data directly to the database server.

Finally, placing the database node is less critical since it will be using either a WiFi or Ethernet network. As such, it merely needs to be on the same network (or made accessible from the network that the data aggregators use).

Data Storage

When considering your data storage, you should consider what the data looks like, that is, what data the sensors produce to include the data type and how the data is used.

For example, if the sensor data is a number in a range of values, say -5.0 to +5.0, the numeric value may not be very meaningful. For example, what does it mean when values are closer to -5.0 versus +5.0? You should always store the original value, but you may want to store a representative value. In this case, there may be several thresholds that determine a qualitative value. Consider the following thresholds for the fictional sensor range. In this case, the value is a voltage reading.

- `-5.0` : Error, no signal

- `-4.9` to `-2.0` : Low

- `-2.0` to `0.0` : OK, decreasing

- `0.0` to `+2.0` : OK, increasing

- `+2.0` to `+4.9` : High

- `+5.0` : Error, no signal
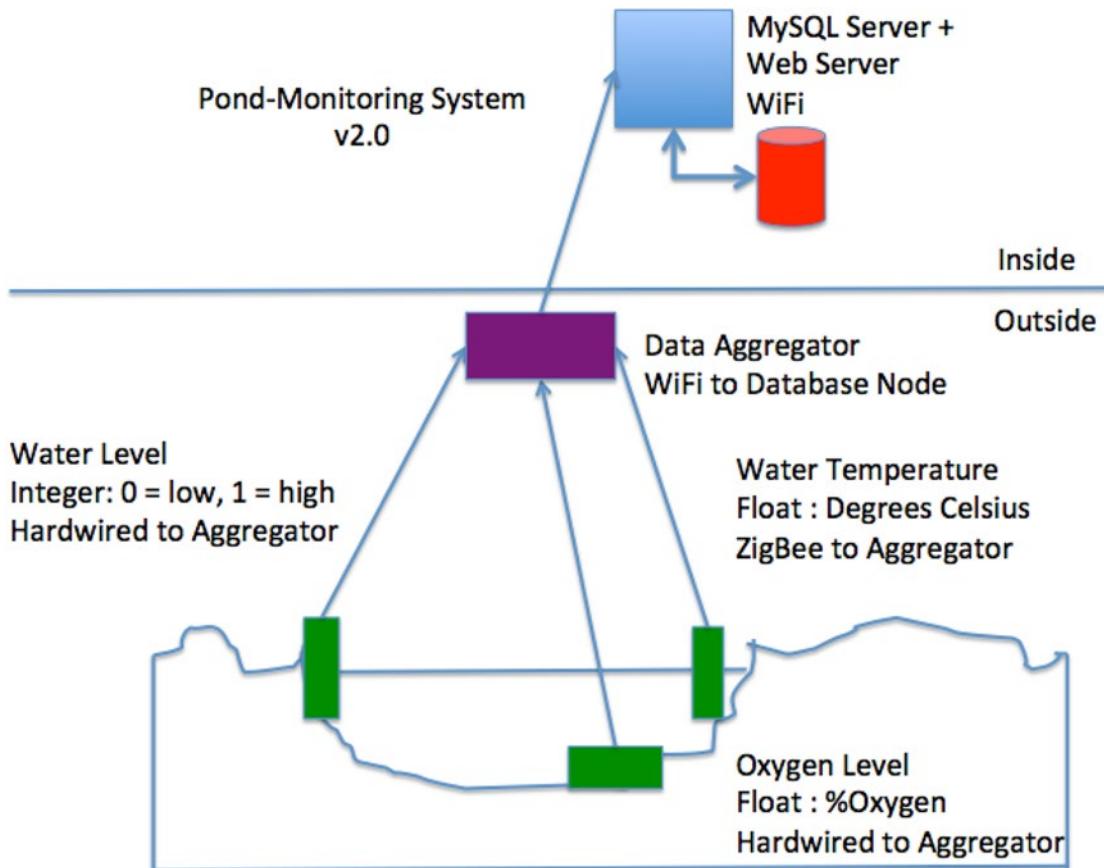
Notice there are two error conditions. I've seen this before in other sensors, and it depends on how the sensor is powered or signaled. While it is not unusual to see a value like this, you are not likely to encounter a sensor that has multiple error readings (but I have seen some).

Notice also that there are four distinct thresholds that tell us what the values mean. For example, if the value read is +3.3, we know the data can be interpreted as "high." Thus, we can store the original value in one column and a category in another. For example, we could have one floating-point field (column) and another text column with values of (low, decreasing, increasing, high, and error). We would use a data aggregator or the database server to assign these values. I show you an example of this in Chapter 5.

This information is vital to planning your data aggregator code. You need to understand what the data means and how best to interpret it. In fact, I recommend documenting it in your code as well as your notebook. The information will be vital should you replace the sensor or discover you need to adjust the thresholds.

For example, if the threshold for increasing or decreasing needs to be adjusted to a higher or lower value, values read that are close to the original threshold may need to be modified. If you had not stored the original value, you would have no way to adjust the data you have already stored.

In addition to how to interpret the data, you should consider how the data flows through your network. I like to make a drawing that shows where each type of data originates and how it moves through the network. Figure 3-2 shows an example of a data flow chart. You can use any form you want—from a simple list written in a log book to a graphical picture written in a structured design language like the Unified Modeling Language (UML).[12]

Figure 3-2. *Data flow chart*

This example is for a pond-monitoring system. I show the data collected in its original form and how the data should be interpreted. This helps me design a database to store the data and decide where to place any code or features that may be helpful interpreting the data.

For example, the water-level detection uses an audible tone generated on the Arduino that hosts the sensor. This allows me to hear a warning when the water level drops (the sensor is triggered). Since I walk by the pond every day on my way to and from work, it is a great example of using multiple cues for the user interface. In this case, the user interface is a web page and the signal generator is on the Arduino.

---

[12]http://uml.org

I also include the communication mechanism. As you can see, two of the data collectors are hardwired to the data aggregator and therefore communicate the data collected via wired connections (I2C in this case). Another data collector uses an XBee (ZigBee protocol) because the water temperature sensor is located too far from the data aggregator (which has to be near the house to transmit data via WiFi to the database server). Finally, notice I use the database server node to host a web server to present the data via a web page.

I recommend using a drawing like this to help you plan your IOT network and even the features for your solution. You really cannot have too much design documentation.

Presentation

When you plan an IOT solution that contains features that permit users to see the data or control actionable devices, you have another level of placement to consider.

This could be rather straightforward if the device is a tablet or computer. In this case, you simply connect it to the same network as the database server and actionable devices. However, if the presentation features are cloud enabled, you may need an intermediate node to isolate your internal nodes from the cloud. This could be a device placed outside the internal firewall, which communicates to the database server and transmits data to a cloud service.

In short, be sure to consider how the data in your database will be presented to the user so that you don't end up with a well-designed data collection mechanism bereft of visualization features. While this sounds obvious, sometimes concentrating on the node placement, data collection, and database design can overshadow how the data will be presented.

Summary

Choosing how to store the data for your IOT solution has many options. You can choose to store the data in the cloud, storing nothing locally on any of the nodes. You could choose to store the data locally in files or memory, building your own storage and retrieval mechanism. Or you could choose to employ a database server dedicated to storing and retrieving your IOT data efficiently and effectively.

Of course, for this book, the assumption is you have chosen or will choose to use a database server to store your data. While you still may choose to cache or even keep a copy locally on some of the nodes, ultimately the database server becomes the focal point for your data.

In this chapter, you examined some of the methods available to you for storing data. You saw examples of how to read and write data in files on the Arduino and Raspberry Pi. I also discussed the benefits, considerations, and recommendations for deploying a database server in your IOT solution. Finally, I discussed some best practices for designing a network of nodes for your IOT solution.

In the next chapter, you'll explore the details of transforming data, from working with data types to normalizing the data, and even talk about how to work with addressing and aggregation. This discussion will prepare you to learn the finer details of working with the MySQL database system.

CHAPTER 4

■ ■ ■

Data Transformation

Whether you have an existing IOT solution, you're developing an idea for a new IOT solution, or you are refining an existing IOT solution, data will be a major focus at some point. Having an excellent IOT solution with near-perfect hardware, communication, and even display capabilities is no insurance against poorly stored or poorly handled data. Getting the data right can be the most difficult task of all. Indeed, I have seen IOT solutions that were good but failed miserably because the data was either not easily presented, understood, or incomplete.

The struggle comes when you consider the type of data being generated and how that data is interpreted. Since most IOT solutions are designed to observe the world around us, it is natural that most of the data generated is sensor data in some form or another. Of course, there could be other data such as images or video from cameras, user input, or even data from social media applications. Fortunately, this data is easily understood. Storage may still be a challenge but typically easily solved.

However, sensor data can be much harder to get right. This is because, as you saw in Chapter 2, sensors can generate analog (for example, voltage) or digital values, which represent the observation. For example, you may find a sensor that you want to use produces values between -5.0 and +5.0. But the sensor may be measuring something like flow rate through a pipe (hose), which you may want to see as liters or gallons per minute. The trick then, is converting the value from the sensor into the expected rate.

There are two classes of transformations that you may need to do with the data. You may need to add additional information such as the units of measure, notes about the events, date or time, and so on. This additional data is a form of annotation. You may also need to perform larger-scale operations on the data where you are combining data from multiple sources (sensors) or even from multiple nodes. These operations are aggregations of the data and can require more effort to implement.

You will discover a number of techniques for handling these and similar transformations in this chapter. The examples are shown from two perspectives: excerpts of code from sketches for the Arduino or Python scripts for use on low-cost computer nodes and suggestions for how to implement the technique in a database. Presenting both techniques for use outside and inside[1] the database server provides you with more breadth of knowledge and thus more tools to develop your IOT solutions.

The database examples will present suggestions rather than concrete examples. The point is to show how it is done in code followed by suggestions for a database implementation so that when you read the next two chapters, you will see how to realize some of these suggestions. I will present many of these suggestions in more detail in Chapter 8. The takeaway then is you will see how much power is available in using a database server.

---

[1]Before database storage and during database storage.

■ **Note**   Some examples may not have database alternatives. In these cases, I mention considerations for storing the data. Also, some of the examples use techniques we have discussed. For those examples, the focus should be on the concepts illustrated.

We begin with a brief discussion and review of what IOT data is and the strategies for making sense of the data, that is, to gain knowledge from the data.

Making Sense of IOT Data

Before we get into examples of how to implement annotation and aggregation of data, let's discuss and review some strategies for making sense of the data. It is important to focus on this topic so that we understand how to make the design of our data storage successful. I present some of the important concepts in the following sections in the form of questions you should ask during the database design.

What Is Being Observed?

The first thing to consider is what you are observing. Don't focus on the data that the sensor is producing (which is also important), but knowing what you are observing will allow you to plan for how the data will be used. I like to make a list of the things I want to observe either from the perspective of the problem or when considering the sensor itself. That is, if I am using a gas detector, I make a list of the gases it can measure and consider how I would use that observation. While the form of that data—the data type from the sensor—may need to be transformed, it is more important to keep the focus on what you are observing. Sometimes this isn't so obvious.

For example, if you are using a sensor to monitor temperature inside an enclosure for a sensitive device, what does the temperature tell you? Let's assume you decided to use the sensor to capture when the temperature reaches a certain threshold because you know that the device cannot withstand temperatures over that threshold. Thus, it seems natural to store those events by saving the temperature measured and perhaps the time and date of the event.

However, what if you want to know whether other temperature ranges can affect the device? In this case, it may be that a certain temperature could be harmful to the device if it were to operate at that temperature over a period of time. Is it then sufficient to simply store the temperature and signal or alert when the temperature reaches the original threshold? It may not be. You may find after some time that there is a correlation with certain temperatures that produces a negative effect on the functioning or accuracy of the device. If this were true, recording the event of exceeding the threshold is not sufficient because the data (temperatures over time) has been lost in favor of storing the event.

Thus, you must consider not only what you are observing but also any other way that the observation could be interpreted and plan what you need to store for those uses. For example, you may find that recording a certain event is sufficient for some use observations, but you may be able to learn more from the data collected over time than a single event.

Is There Another Way to Make the Observation?

Closely related to what you expect to observe is how to make the observation. Sometimes it may not be possible (or too costly) to make a direct observation with certain sensors. For example, suppose you have a garden pond and want to determine when the filters need cleaning.

In some garden ponds, filters are used to remove debris floating in the water (such as leaves, waste, and so on). Over time, these filters may become full of debris and thus lose the ability for water to flow through the filter. Further, let's assume there is no easy mechanism to measure the flow of water passing through the

filter. That is, the filters are inside an enclosure that houses the pump, and thus the filters clean the water prior to it flowing into the pump well. Further, there is no sensor designed to measure the flow rate of a filter in the enclosure. How would you make the observation that the filters need cleaning?

If you have owned and maintained a garden pond using a filter system like this, you would know that there is a causal effect you can observe to determine whether the filters need cleaning. More specifically, as the filters become full of debris, the water in the pump well lowers (lower flow, less water in the reservoir). Thus, you could determine that when the water level in the pump well is reduced to a certain level, the filters need cleaning.

However, this observation is not ideal because it is also possible for the water level in the pump well to become low if there is a leak in the pond or if there is significant evaporation. Thus, we must add these events to the list of possible things we can learn from the observation. In this case, we have one primary and two secondary things we can learn from the observation of water level in the pump well.

KNOW YOUR DOMAIN

This brings up a very good point. If you were asked to make an IOT solution for a garden pond but did not know or have any experience with the subject matter, you may not be able to understand how to observe the state of the filters. Thus, it is important to know or study the domain within which you are working so that you can explore alternative methods of making observations.

■ **Tip**   You should consider alternative methods of making an observation including observing the causal events rather than the actual or physical phenomena.

How Often Do You Need to Record the Observation?

You should also consider the frequency you want to record the observations, that is, how many times you want to record the values from the sensor. Some sensors may have a timer circuit or a minimal threshold for when a measurement can be taken. Simple sensors such as switches can be instantaneous, whereas sensors such as gas, water salinity, or oxygenation may have a significant threshold, reducing the number of observations you can make in a given time frame.

It is also possible that the frequency of the observation may have little or no bearing on the knowledge you expect to gain. For example, if we want to measure temperature in a building or room, can we learn anything if the temperature values were recorded once every three seconds? How about twice a minute? What about once an hour or once every six hours?

The answer depends on what we want to learn. Yes, we're back to that again. More specifically, do we want to be able to track when the temperature changes down to a specific minute, hour, or time of day? Moreover, do we want to perform any analysis on the rate of change?

For example, if we are measuring temperature in a building or room that does not have a controlled climate (for example, a barn or covered bridge), are we interested to see how the temperature changes over time? If we are, how accurate do we want the data to be? That is, do we want to be able to detect how quickly the temperature changes? If we do, storing measurements taken more frequently may permit more accurate detection of these changes as well as more accurate rates of change. On the other hand, if we just want to be able to determine average temperatures at different periods during the day (for example, morning, noon, night), measurements taken every few hours may be sufficient.

Thus, we must consider what we want to observe as well as whether changes over time are beneficial. That is, measurements taken using short intervals can detect changes and trends more accurately than measurements taken at larger intervals. Indeed, it is possible if the interval is large enough that a fluctuation in the measurement would be missed.

■ **Tip**   Consider how often you want to record the observation based on what you may learn from change over time.

What Type of Data Does the Sensor Produce?

The next thing to consider is the type of data that the sensor is producing. Resist the temptation to jump to how the data is interpreted—we'll do that next. For now, make a note of what the data type is so that you can refer to it when you implement your solution.

It is possible the data type of the data from the sensor could be quite different from what we are observing. We have already seen this for sensors that produce analog data in the form of voltage. A voltage of plus or minus volts may not be obvious as to what it indicates. Making a note of what data each sensor produces will help you when it comes time to write code to read, analyze, and store the observations.

■ **Tip**   Make a list of the data types that your sensors produce. Keep this list handy when designing your solution.

Are There Interpretations Needed for the Observation Data?

Once we know what we are observing, what we can learn from the observation, how frequently we need to make the observation, and what the data type is, we now must consider if there are any interpretations that need to be made.

Returning to the analog sensor discussion, most analog sensors have specific interpretations, and indeed the interpretation is documented. That is, the documentation for the sensor will tell you how to interpret the values produced. This is presented via a common mechanism called a *data sheet*.

READ THE DATA SHEET

You can find more about how to interpret the data from the data sheet for the sensor. Most manufacturers produce a short description of their device including operating parameters (voltage, current, and so on) as well as the data type the sensor produces and how to interpret that data. Refer to the data sheet if you are unfamiliar with the sensor or its discrete components.

Once you understand how the data type is interpreted, make notes about how to perform the interpretation. I like to include this data in a notebook along with pseudo-code for how to make the observation. Not only will it help you write code to interpret the data, it can also help you should you consider including the interpretation inside the database.

Additionally, you should also consider how the data is to be presented, that is, how you want to show the data to the user. For example, it would not be helpful for the user to read a value of -4.12 volts, but it would be helpful for the user if that value was presented as category such as "ok," "wet," or "dry."

■ **Tip**   Make a plan for how to interpret and present the data. Record it in a notebook for reference.

What Level of Accuracy Do You Need?

It may come as some surprise, but sensors are not always 100 percent accurate. In fact, most sensors that are marketed for hobbyists and enthusiasts are only about 90 to 95 percent accurate. That is, 5 to 10 times out of 100, the sensor will not produce an accurate value.

If your sensors are only 95 percent accurate, you can only expect your data to be the same. Thus, you must determine the accuracy required for your solution when choosing or analyzing data from your sensors. Fortunately, if you need a higher level of accuracy, you can typically find sensors to match your expectations. However, in my experience, the more accurate the sensor, the more expensive they become.

■ **Tip**   Balance accuracy of your sensors with expected accuracy of your data.

What Is the Lifetime of the Data?

Finally, consider how long you want or need to keep the data. Some IOT solutions, especially those that use the cloud,[2] seem to keep the data for only a few hours, days, or weeks. Not only does this seem arbitrary, but it also ensures you may never be able to perform any data analysis on historical data. That is, you may lose knowledge because your sample set is too small.

When considering the lifetime of the data, you should consider all the previous questions and make a plan for how long you want to keep the data, more specifically, whether data analysis of data over longer periods of time will be beneficial.

One factor to consider is if larger amounts of data have an effect on the efficiency of your solution. This could be a case where more data makes searches or even certain code functions slower because it has to spend more time reading and comparing data. It could also be possible that your storage solution has a limit to how much data can be stored. Thus, you should consider hardware and software limitations when determining data lifetime.

Once you determine a lifetime of the data, you should then determine whether you need the data accessible or not. If you do not need it accessible from your solution or its application, you can consider removing the old data for safe keeping such as a file archive or backup. If it still must be accessible, you may want to consider using an alternative or supplemental storage mechanism. For example, if you are storing the data in a file, you can simply open a new file.

ARCHIVE, DON'T DELETE

If you determine you can purge data older than a certain date, don't delete the data. Instead, archive it so that if you ever need to access the older data, you can. For databases, a database backup may suffice or you can use a table of the same schema to store older data. For file-based storage, make a copy of the file on removable media or another device.

Once you have a plan for the lifetime of the data and what to do with the older data, record this plan in your notebook so that you can write the code or implement the correct processes to periodically purge the data.

---

[2]Not always the case, but it is a trend I've noticed.

> ■ **Tip** Make a plan for how long you want to store the data. Implement any purge events as backup or archival rather than deletion.

Now that we've seen some things you need to consider about making sense of the data, let's see several techniques for implementing these strategies starting with annotation.

Some of the code examples in the following sections are written for file-based storage. Other media forms are similar, but the actual write methods may differ. I also simulate the collection of data from sensors. You would typically use a method written to retrieve the data in a similar manner.

I also include considerations for using the database to add the annotations or implement the aggregation. I use these sections to introduce the database concepts to prepare you for the more in-depth tutorial on MySQL in the rest of this book.

Let's begin with a look at some of the more common annotations you may want to make.

Annotation

Annotating data for your IOT solution is simply any additional data you add to, combine with, or calculate from the data. For example, you may want to store a string to describe the event, sensor, node, and so on. Or you may want to perform some transformation on the data and save it along with the original data.

> ■ **Tip** It is always a good practice to save the original values.

I demonstrate a number of these transformations in this chapter, providing a code example that you can use on your data aggregators or data nodes such as those hosted by an Arduino, Raspberry Pi, or similar low-cost computer node. Thus, where appropriate, I present both an Arduino sketch excerpt and an example using Python. While you may not be familiar with both languages, you can use these examples to help guide you to write in your chosen language.

CAN YOU HAVE TOO MUCH ANNOTATION?

Annotation should be for the benefit of the user or to make data more informative. Thus, you must consider whether the annotation will add anything, but more importantly, the annotation should not complicate or otherwise obscure the data. Some of the examples in this chapter get close to that line of benefit versus detriment, but this is for illustrative purposes. Use your own judgment as to how much annotation is enough for your needs. Too much and you risk obscuring knowledge and too little can make the data harder to use.

Now that you know what annotation is, let's see a few examples beginning with the easiest forms of transformation—adding text and recording notes.

> ■ **Note** The following code examples are excerpts for brevity. Complete sketches or scripts are available from the book source code download site.

Recording the Sensor Name or Adding Notes

The simplest form of annotation is adding a short string to the data stored. You may want to do this to ensure the values stored are understood such as what event is being observed or which data collector (sensor) generated the data. You may also want to add any notes to the data stored such as user input or maybe subjective observations such as "cat slept in the planter." The added data could be helpful when reading the data at a later date, especially if you are writing the data to a file as either a log or data storage.

However, you must use this technique with some care. Adding too much text could make the data harder to interpret or perhaps harder to parse should you need to do so in the future. Since adding strings is easy, I will present some examples with a short explanation.

Code Implementation

Adding text to data stored in a file is not difficult. Listing 4-1 shows how to add text to a row of data in an Arduino sketch. In this case, I add the data from two (simulated) sensor readings along with an explanation of each. I also add statements for debugging (the Serial.print statements).

Listing 4-1. Simple Annotation (Arduino)

```
/**
  Example of simple annotation.

  This project demonstrates how to save data to a
  microSD card as a log file with sting annotation.
*/
#include <SPI.h>
#include <SD.h>
#include <String.h>

// Pin assignment for Arduino Ethernet shield
//#define SD_PIN 4
// Pin assignment for Sparkfun microSD shield
#define SD_PIN 8
// Pin assignment for Adafruit Data Logging shield
//#define SD_PIN 10

File log_file;  // file handle
String strData;  // storage for string

// Simulated sensor reading method
float read_sensor(int sensor_num) {
  if (sensor_num == 1) {
    return 90.125 + random(20)/10.00;       // sensor #1
  } else if (sensor_num == 2) {
    return 14.512313 + random(100)/100.00;  // sensor #2
  } else {
    if (random(25) >= 5) {
      return (float)random(14)/10.00;
    } else {
      return -(float)random(14)/10.00;
    }
  }
}
```

107

```
void setup() {
  Serial.begin(115200);
  while (!Serial);
  Serial.print("Initializing SD card...");
  if (!SD.begin(SD_PIN)) {
    Serial.println("ERROR!");
    return;
  }
  Serial.println("ready.");

  if (SD.remove("data_log.txt")) {
    Serial.println("file removed");
  }

  // initiate random seed
  randomSeed(analogRead(0));
}

void loop() {
  delay(2000);
  log_file = SD.open("data_log.txt", FILE_WRITE);
  if (log_file) {
    Serial.println("Log file open.");
    strData = String("Temperature sensor: ");
    strData += read_sensor(1);
    strData += " ";
    delay(1000);
    strData += ", Barometric sensor: ";
    strData += read_sensor(2);
    log_file.println(strData);
    Serial.print("Data written: ");
    Serial.println(strData);
    log_file.close();
    Serial.println("Log file closed.");
  } else {
    Serial.println("ERROR: Cannot open file for reading.");
  }
}
```

In this example, I use the String class to concatenate the strings with the sensor readings. While there are other methods, this one seems clearer to demonstrate the annotation. Notice I place a comma in the string before the second sensor. This comma can be used to help parse the data should you need to do so in the future. For example, you can split the data on the comma and then locate the data values at the end of the string after the colon. Here is a sample of the output or in this case an excerpt from the log file:

```
Temperature sensor: 90.82, Barometric sensor: 15.00
Temperature sensor: 91.43, Barometric sensor: 15.09
Temperature sensor: 91.13, Barometric sensor: 15.23
```

While this example shows how to save the sensor name with the data, it would be equally easy to add more text on the end of the line of output by simply appending the string. For example, you could add a note at the end of the string as follows. Here I've added a method, which returns the user's input from a prompt (or web form).

```
...
strData += ", Barometric sensor: ";
strData += read_sensor(2);
strData += " Notes: ";
strData += read_user_input();
log_file.println(strData);
```

To implement a similar annotation in Python, you would use code such as shown in Listing 4-2. Here, I reproduce the results that the Arduino code generated. That is, I write the same data, but I use a completely different method. To help understand some of the formatting, I include the entire code.

Listing 4-2. Simple Annotation (Python)

```
from random import random, randint
import string

def read_sensor(sensor_num):
  if (sensor_num == 1):
    return 90.125 + random()*10  # sensor #1
  elif (sensor_num == 2):
    return 14.512313 + random()  # sensor #2
  else:
    if (randint(0,25) >= 5):
      return randint(0,14)/10.00
    else:
      return -randint(0,14)/10.00

strData = "Temperature sensor: {0:.2f}, Barometric sensor: {1:.2f}\n"
file_log = open("data_log_python.txt", 'a')
file_log.write(strData.format(read_sensor(1), read_sensor(2)))
file_log.close()
```

Here you see an example of string formatting using the format() method. Notice how I use formatting codes as presented in the string named strData to limit the output to two decimal points such as {0:.2f} and {1:.2f}. These codes tell the format() method to substitute the first and second parameters (read_sensor(1) and read_sensor(2)), formatting the data as a floating-point number with two decimal places.

While this example was basic and meant to be an example, most developers would likely not save the name of the event or sensor with every row. What is more likely is you would write the name of the sensors as a header line at the beginning of the file. This way, you always know what the columns of data mean. In fact, this is precisely how you would use annotations like this in a database solution.

Database Considerations

Annotation with text is easy to do in the database. If we want to include the name or type of the sensor, we can simply name the column of the table accordingly. In this way, the columns names describe the data. For example, if we had a table to store the rows from the examples in this section, the table may look something

like Listing 4-3. Don't worry about knowing what all the parts of the command are; just focus on the readability of the example.

Briefly, the table is the database construct we use to store the data. It forms the layout of the data. We will learn more about creating tables in the next chapter. For now, notice the columns. Here we see there are two columns that are named to correspond with the data we are collecting. Indeed, if we were to enter the data from the previous example, it would look like the results at the end of the listing. I use a SELECT statement (the method for retrieving rows from the database) to show sample entries.

Listing 4-3. Sample Database Table and Sample Results

```
CREATE TABLE `simple_annotation_test` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `temperature` float NOT NULL,
  `barometric` float NOT NULL,
  `notes` char(128) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

mysql> SELECT * FROM simple_annotation_test;
+----+-------------+------------+-------+
| id | temperature | barometric | notes |
+----+-------------+------------+-------+
|  1 |       90.82 |         15 | NULL  |
|  2 |       91.43 |      15.09 | NULL  |
|  3 |       91.13 |      15.23 | NULL  |
+----+-------------+------------+-------+
3 rows in set (0.01 sec)
```

Notice there is also a column for notes. This shows that we can also handle saving any additional notes or user input along with the samples taken. To do so, we include it when we save data.

DESCRIPTIVE COLUMN NAMES

This is one of my pet peeves about poor database design habits. That is, you should always use descriptive names when naming objects (tables), columns, indexes, and so on. That doesn't mean you need to use 200 characters for each column name (that would be absurd), but you should avoid the temptation of using single character names such as a, b, c, and so on.

As you can see, annotation of data in the database is accomplished through either naming the columns, creating columns to store text, or even creating special data types to help annotate the data. For example, you can define a column to contain a set of values—called *enumerations*—such that we can specify the numeric value of the enumeration when we save data. In fact, there are many ways we can do annotation in the database. We will see more examples in later sections.

Recording the Date and Time

Aside from adding text to the data in the form of column names, notes, and so on, we often need to store the date and time when an event or series of events are observed. That is, we want to save when the sensor was read. However, most microcontroller boards and even some low-cost computer boards do not have a real-time clock (RTC) circuit. For example, the Arduino does not have an RTC nor does a Raspberry Pi.
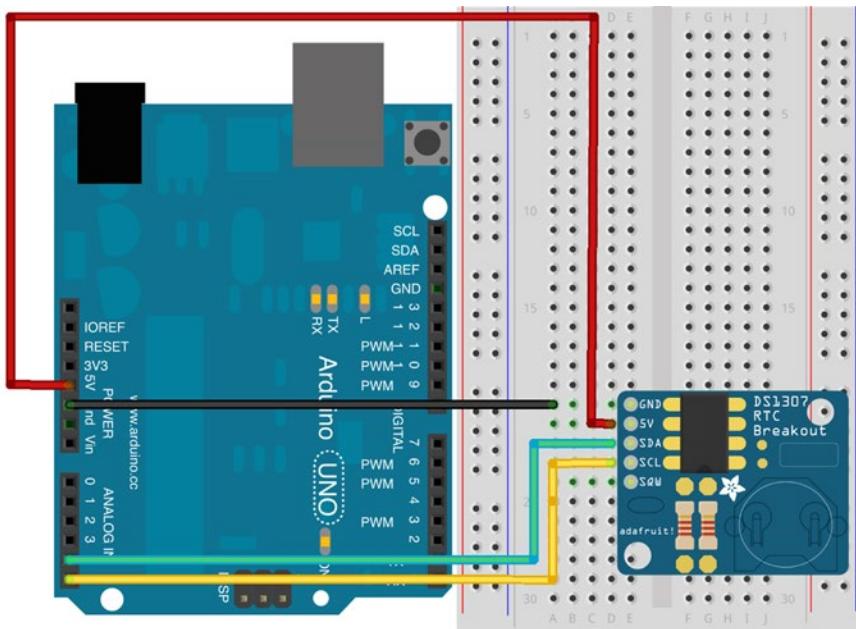
An RTC is needed to ensure accurate time keeping because the clock (a special crystal or similar mechanism that generates a pulse) that is used to advance instructions in the processor (microcontroller) does not pulse or cycle at a frequency that can be used to accurately calculate time. Furthermore, RTC circuits have a battery to power a small amount of memory to store a value for the time (sometimes seconds since a specific epoch). Thus, boards without an RTC must be either programmed with a starting date and time or instructed to get the date and time from a time server on the Internet.

Fortunately, there are several excellent products that perform well and include an onboard battery that powers the clock even when the board is powered down. Adafruit's DS1307 Real Time Clock breakout board kit (`www.adafruit.com/products/264`) is an outstanding module to add to your project. Sparkfun also has a product named Real Time Clock Module (`www.sparkfun.com/products/99`) that uses the same DS1307 chip and interface as the Adafruit offering. You can use either with your Arduino, Raspberry Pi, or any board that has an IC2 interface.

Using a Real-Time Clock Module

The RTC module uses an I2C interface that is easy to connect to the Arduino. Simply connect 5V power to the 5V pin, ground to the GND pin, the SDA pin-to-pin 4 on the Arduino, and the SCL pin-to-pin 5 on the Arduino. Figure 4-1 shows the wiring diagram for connecting the RTC module to an Arduino.


Figure 4-1. *Arduino with an Ethernet shield and RTC module*

■ **Note**   If you are using the Leonardo board, you have to use the new SCL/SCA pins located next to AREF, GND, and pin 13 nearest the USB connector.

There is a bit of code needed to initialize the RTC, but it isn't difficult. First, we must download the RTC library. You can find an excellent library to use at https://github.com/adafruit/RTClib. Download the library by clicking the Download Zip button, unzip the archive, and then copy the folder to your Arduino/ Library folder. You may need to rename the folder. I recommend using RTCLib for the folder name.

■ **Tip**   If you copied the library while the Arduino IDE was open, you may need to restart the IDE for it to recognize the library.

Once the library is ready, we can add the code to use the library. We need to include the library header and define an instance of the RTC class. The following are the lines of code needed:

```
#include "RTClib.h"
...
RTC_DS1307 rtc;  // real time clock
```

Next, we need to initialize the RTC class with the begin() method, and if the module has not been used before or if you have changed the battery, we also need to set the time with the adjust() method. The following is the code you can use to accomplish both of these tasks:

```
...
  rtc.begin();
  if (!rtc.begin()) {
    Serial.println("Couldn't find the RTC module.");
    while (1);
  }

  if (!rtc.isrunning()) {
    Serial.println("ERROR: The RTC module is not working.");
  } else {
    // Comment out this line after the first run, it is only needed for setting
    // the date and time for the first time.
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }
```

Notice the parameters passed to the method rtc.adjust(). This converts the values of the current date and time to a DateTime class, which the method needs to set the date and time for the module. The __DATE__ and __TIME__ are macros that retrieve the date and time from your system at the point when the sketch is compiled. Thus, you need to call this method only when you first start using the RTC. You do not need to call it every time you run your sketch. If you do, you will be setting the RTC to the same values every time. This is not what you want. Thus, I've written the code so that you can comment out the method after the first compilation.

Now let's see how to use the RTC in a sketch to capture the date and time on an Arduino.

Code Implementation

The code to add the date and time to the row written to the file requires a bit more work than you may expect. The RTC module provides primitives for getting the separate elements of a date and time such as month, year, hour, and so on. Thus, we need to get these values and combine them into a format we expect to see. In this case, we want to see a format of month/day/year hour:minute:seconds. Listing 4-4 shows code to get and format the date and time.

Listing 4-4. Date and Time Annotation (Arduino)

```
String get_datetime() {
  DateTime now = rtc.now();
  String dateStr = String(now.day());
  dateStr += "/";
  dateStr += now.month();
  dateStr += "/";
  dateStr += now.year();
  dateStr += " ";
  dateStr += String(now.hour());
  dateStr += ":";
  dateStr += String(now.minute());
  dateStr += ":";
  dateStr += String(now.second());
  return dateStr;
}

void loop() {
  delay(2000);
  log_file = SD.open("data_log.txt", FILE_WRITE);
  if (log_file) {
    Serial.println("Log file open.");
    strData = String("Temperature sensor: ");
    strData += read_sensor(1);
    strData += " ";
    delay(1000);
    strData += ", Barometric sensor: ";
    strData += read_sensor(2);
    strData += " ";
    strData += get_datetime();
    log_file.println(strData);
    Serial.print("Data written: ");
    Serial.println(strData);
    log_file.close();
    Serial.println("Log file closed.");
    delay(2000);
  } else {
    Serial.println("ERROR: Cannot open file for reading.");
  }
}
```

As you can see, the code is nearly the same as the previous example. The only difference is the addition of the method named get_datetime() to get and format the date and time returning a string, which we then append to the row being written to the file. Sample rows from this code are shown here:

```
Temperature sensor: 92.33, Barometric sensor: 15.23 Datetime: 11/07/2015 22:36:32
Temperature sensor: 90.72, Barometric sensor: 15.32 Datetime: 11/07/2015 22:36:48
Temperature sensor: 94.38, Barometric sensor: 15.13 Datetime: 11/07/2015 22:36:50
Temperature sensor: 96.74, Barometric sensor: 14.95 Datetime: 11/07/2015 22:36:50
```

If you want to do something similar in Python and your platform has an RTC on board, the code is rather easy. You simply add the date and time as shown in Listing 4-5. I include the entire code here so you can see the supporting code for reading the date and time.

Listing 4-5. Date and Time Annotation (Python)

```python
from datetime import datetime
from random import random, randint
import string

def read_sensor(sensor_num):
  if (sensor_num == 1):
    return 90.125 + random()*10  # sensor #1
  elif (sensor_num == 2):
    return 14.512313 + random()  # sensor #2
  else:
    if (randint(0,25) >= 5):
      return randint(0,14)/10.00
    else:
      return -randint(0,14)/10.00

def get_datetime():
  return datetime.strftime(datetime.now(), "%m/%d/%Y %H:%M:%S")

strData = "Temperature sensor: {0:.2f}, Barometric sensor: {1:.2f} Datetime: {2}\n"
file_log = open("data_log_python.txt", 'a')
file_log.write(strData.format(read_sensor(1), read_sensor(2), get_datetime()))
file_log.close()
```

Notice we used a method to get and format the date and time in a similar way as we did in the Arduino code. However, Python has more advanced methods (borrowed from C, of course) that we can use to format the date and time using a string containing format codes. Notice the get_datetime() method. We get use the method from the datetime class named strftime() to create a string from the current date and time from the datetime.now() method. The format string is then used by the strftime() method. The output of this code is similar to the Arduino code.

```
Temperature sensor: 92.33, Barometric sensor: 15.23 Datetime: 11/07/2015 22:36:32
Temperature sensor: 90.72, Barometric sensor: 15.32 Datetime: 11/07/2015 22:36:48
Temperature sensor: 94.38, Barometric sensor: 15.13 Datetime: 11/07/2015 22:36:50
Temperature sensor: 96.74, Barometric sensor: 14.95 Datetime: 11/07/2015 22:36:50
```

If your low-cost computer board does not have an RTC, you may be able to add one if the board has an I2C interface and sufficient support in the operating system for the RTC. For example, you can add an RTC to the Raspberry Pi. Adafruit has an excellent tutorial for this at https://learn.adafruit.com/adding-a-real-time-clock-to-raspberry-pi/overview. Once you have added the clock, your Python (or other language) scripts will get the correct date and time from the operating system primitives.

Database Considerations

Saving a date and time in a row for a database is possible in one of two ways: you could add a column to the table of the type `datetime` and provide the date and time in your code that communicates with the database server (you issue an INSERT statement to add data to the table), or you could use a `timestamp` column, which is a special column that the database server populates for you when the row is inserted. Let's look at each of these options.

You can add a date and time column to the table by specifying the `datetime` data type. This value is added or updated as you would any other column. Listing 4-6 shows an example table schema that includes a single date and time column. I highlight the column for easier reading.

Listing 4-6. Database Table with a datetime Column

```
CREATE TABLE `date_annotation_test` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `temperature` float NOT NULL,
  `barometric` float DEFAULT NULL,
  `date_saved` datetime DEFAULT NULL,
  `notes` char(128) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1
```

Using a `datetime` column requires you to supply the value in a format that is compatible for MySQL. Fortunately, the same format we used so far works well (others do too—see the MySQL online reference manual for more examples). The following shows how we can save a row to the table, providing the date and time value for the new column.

```
INSERT INTO date_annotation_test VALUES (null, 91.34,15.04,'11/7/15 23:16:30', 'Test
datetime.');
```

Now let's see an example of the output from this table with sample data added. In this case, I added several rows specifying the date and time for each row.

```
mysql> SELECT * FROM  date_annotation_test;
+----+-------------+------------+---------------------+---------------+
| id | temperature | barometric | date_saved          | notes         |
+----+-------------+------------+---------------------+---------------+
|  1 |       90.82 |         15 | 2011-07-15 23:15:10 | NULL          |
|  2 |       91.34 |      15.04 | 2011-07-15 23:16:30 | Test datetime.|
+----+-------------+------------+---------------------+---------------+
2 rows in set (0.00 sec)
```

The other form of adding date and time annotation is using a `timestamp` column. This is a special data type that is updated automatically by the database server. We can use only one `timestamp` column per table. The following shows an excerpt from a table CREATE statement. I added this column to the previous example with the following ALTER statement.

```
ALTER TABLE date_annotation_test ADD COLUMN data_changed TIMESTAMP AFTER notes;
```

Notice the additional options after the DEFAULT option. These are added by default, and it shows that the value is changed whenever the row is created or updated. So, yes, the timestamp column is not fixed once you set it. Let's see an example of adding a row with the new column.

```
INSERT INTO date_annotation_test VALUES (null, 91.34,15.04, null, 'Test timestamp.',null);
```

Here I pass in the NULL value, which tells the database server to use the default, and as we saw in the earlier column definition, that is an update to the column. Notice also I left the date and time column (date_saved) null, meaning do not update that column. Thus, this row should show no value for the date and time column, but the date and time that the row was inserted for the timestamp column (data_changed). The following shows the cumulative data:

```
mysql> SELECT * FROM  date_annotation_test;
+----+-------------+-----------+---------------------+----------------+---------------------+
| id | temperature | barometric | date_saved         | notes          | data_changed        |
+----+-------------+-----------+---------------------+----------------+---------------------+
|  1 |       90.82 |        15 | 2011-07-15 23:15:10 | NULL           | 2015-11-08 15:55:50 |
|  2 |       91.34 |     15.04 | 2011-07-15 23:16:30 | Test datetime. | 2015-11-08 15:55:50 |
|  3 |       91.34 |     15.04 | NULL                | Test timestamp.| 2015-11-08 15:57:27 |
+----+-------------+-----------+---------------------+----------------+---------------------+
3 rows in set (0.00 sec)
```

Notice the last two rows have the same data. I did this intentionally so you could see the difference between adding a row with a datetime column and adding a row with a timestamp column.

Thus, using a timestamp column means the database does the work for you, so you can keep track of when the data was added or changed. However, there are limitations to using the timestamp data type. First, you can have only one per table. Second, the values will be overwritten on any change to the row. Thus, if you want to save a specific date and time but later want to update the row (say to add notes), you may either want to reconsider using the timestamp or add one or more datetime columns for storing static date and time values.

But wait, why are the data_changed values the same for the first two rows? You may have noticed that the data_changed column for the first two rows have the same value. This is because I added the new column to an existing table. Internally, the database server must update all rows in order to accommodate the change, and thus each row that already exists in the table gets changed; and since the column uses the ON UPDATE CURRENT TIMESTAMP (by default), the new column was updated. Keep this in mind when changing the schema of an existing table.

■ **Tip**   You should consider the effects of adding a timestamp column to an existing table as well as how the timestamp column is updated to ensure you are saving the data you want.

Data Type Transformations

Changing the type of a data element is something you should take some time to think about and plan carefully. This is because it is easy to lose data if you do not perform the conversion properly. This goes beyond changing a float to a string and back (although you can have issues there too if you strip off too many decimals). You must consider the amount of storage required as well as range of values. For example, an

integer and an unsigned integer may be the same size[3] (2 bytes), but the range of values is different because an integer allows for sign-limiting values (+/-) to the range -32768 to 32767 while an unsigned integer stores values in the range 0 to 65535.

You also need to consider precision for the data. Floating-point numbers are real numbers with a number of decimals. Most platforms state quite clearly that floating-point data types are estimated and not exact—again, because of the decimal portion. Thus, you must consider this when converting data types. For example, if you change a floating-point number to an integer, you may have rounding issues. Of course, changing an integer to a float may also introduce rounding issues. Similarly, changing a type from a larger storage (number of bytes) to a smaller storage (fewer bytes), you run the risk of overflowing the destination memory and generating an invalid value.

Another consideration is performing arithmetic. That is, the resulting value or intermediate values must "fit" in the range of values for the data type. For example, if you add two unsigned integers and assign the result to another unsigned integer, you must ensure the value is in the range 0 to 65535. Otherwise, the value may overflow and cause a number of issues, most notably an incorrect value.

■ **Note**　Some programming languages have strict type checking and can detect or warn when overflows are possible, but this is no substitute for careful programming (using the correct data type).

You may be wondering what overflow is. Observe the following short Arduino sketch. Here I have two variables, a long and an unsigned integer starting with a value of 65535. Watch what happens.

```
void setup() {
  unsigned int uint_val = 65535;
  long long_val = 65535;
  Serial.begin(9600);
  while (!Serial); // Wait until Serial is ready - Leonardo
  Serial.print("1> ");
  Serial.print(uint_val);
  Serial.print(" ");
  Serial.println(long_val);
  Serial.print("2> ");
  uint_val++;
  long_val++;
  Serial.print(uint_val);
  Serial.print(" ");
  Serial.println(long_val);
  Serial.print("3> ");
  uint_val++;
  long_val++;
  Serial.print(uint_val);
  Serial.print(" ");
  Serial.println(long_val);
}

void loop() {
}
```

---

[3]I say may be the same size because variable size is dependent on the processor and platform. It is best to check the documentation for your platform before considering memory allocations for data types.

And here is the output (from the serial monitor):

```
1> 65535 65535
2> 0 65536
3> 1 65537
```

I print the values on the first row of output and then add one to each and print them again on the second row. Notice what happens when I add 1 to the variables and print them. You may expect the unsigned integer to be 65536, but it's 0 because there was an overflow where the number of bits to represent the number exceeded the maximum size of the variable (2 bytes). The value of 65536 requires 17 bits (10000000000000000), and there are only 16 in two bytes and the first 16 are taken starting on the right (0000000000000000). What may seem alarming is there was no warning of an overflow. The third row shows what happens when I add 1 again but we see the unsigned integer variable has reset.

Thus, you should strive to use the smallest data type possible, especially when working with microcontrollers or other devices with limited memory. To exercise this practice, you need to understand the range of values that each data type can safely store. To get an idea of what it could mean for the Arduino platform, Table 4-1 lists some common data types, memory size, and range of values. Notice how much larger the floating-point variables are than the integers.

Table 4-1. *Arduino Data Types*

Data Type	Num Bytes	Range of Values
boolean	1	Logical true/false
byte	1	Unsigned number from 0 to 255
char	1	Signed number from -128 to 127, typically ASCII characters
unsigned char	1	Character values 0 to 255
word	2	Unsigned number from 0 to 65535
unsigned int	2	Unsigned number from 0 to 65535
int	2	Signed number from -32768 to 32767
unsigned long	4	Unsigned number from 0-4,294,967,295
long	4	Signed number from -2,147,483,648 to 2,147,483,647
float	4	Signed number from -3.4028235E+38 to 3.4028235E+38

There is one more aspect to using the smallest data types possible. Some data types will cause your sketch to run slower. For example, using floats in arithmetic operations can be as much as 16 times slower than using integers. If precision is not an issue and performance is, you may want to consider rounding floating-point values to integers (hint: multiply by 100 to preserve 2 decimal points prior to the conversion). Just be sure to use the largest integer you can.

Rather than present a set of code examples, the following section discusses some of the finer points of converting and using data types.

Code Implementation

While most experienced programmers would caution against converting data types unless absolutely necessary,[4] there are a few cases where it is possible to justify if done carefully. For example, if you are working with a language that has loose type checking and want to do some arithmetic resulting in a floating point value but unintentionally use integers in the equation, you may get an integer result. To overcome this, you may need to change the integers to floating-point numbers with a special operation called a *cast* (also called *casting*). Casting in Arduino C is implemented by placing the new data type in parentheses. An example is shown here:

```
int shift_value = 100;
val_shifted = (float)shift_value * 95.0675;
```

Another argument for using certain data type conversions is to save memory. This is most common when working with microcontrollers and other processors that have limited memory for variables. Those new to working with devices with limited memory may not think saving a byte here or there could make a difference, but it really can make a huge difference.

For example, if your sketch uses a lot of variables for calculations, you may need to consider reducing the types and therefore the sizes of your variables. For example, consider how many bytes a relatively small matrix of 20×20 values would consume if each cell were a double. Yes, that's 20×20×4 = 1,600 bytes. If your Arduino has only 2Kb of memory, you're likely to encounter problems.

If you encounter a problem like this and the range of values you are working with is smaller than the data type you've chosen, you may be able to convert them. For example, if the values in the matrix never exceed about +/-320 and precision permits the use of only 2 decimals, you can convert the values to integers as follows:

```
int new_val;
new_val = (int)(orig_val * 100.00);
```

This code results in a value such as 222.66 to 22266, which is small enough to fit in an integer data type and thus can save 800 bytes in memory. This is quite a large savings for an older Arduino board! However, I should note here that the values would be truncated, not rounded. So while you can get the data back to a float by dividing by 100, you will not recover any additional precision of the original value.

Another possible data type conversion you may encounter is converting bytes to characters or retrieving larger numeric values from a series of bytes. This is likely to come about when communicating between devices. That is, most communication mechanisms send data in a byte stream. In this case, we may need to extract data such as integers, floating-point numbers, and even character strings (text) from the byte stream.

Converting byte to characters is simple since they are treated the same in code (each is one byte). However, they can be interpreted quite differently. For example, an ASCII character uses only the first seven bits, while a byte is treated as eight bits.

If you have to extract numbers from byte streams, you will need to know how the values are encoded. For example, a small value that fits in a single byte will require, well, one byte. A larger value may be an integer and require two bytes, and a long integer may require four bytes. Aside from the number of bytes, you must also know in what order the individual bytes are stored. Most microprocessor and microcontroller platforms store them with the smallest byte first.[5]

---

[4]My own experience and training force me to maintain strict type adherence, but I too admit there are some cases where it may be necessary. That is, beyond the trivial.

[5]Called *little-endian*. See endianness (https://en.wikipedia.org/wiki/Endianness).

Thus, to retrieve a value from the byte stream, you must read one byte at a time shifting each byte into position. That is, you read the first byte and then the second, shifting it to the left and adding the first byte. Think of it this way: there are two bytes in an integer and you are splitting them. To reconstruct the integer, you have to take the first byte (the leftmost bits) and shift it 8 bits to the left. This leaves the rightmost byte empty. An illustration follows. I use hexadecimal values to keep the example easy to read.

```
Integer: 0x5AFE
Left-most byte: 0x5A
Right-most byte: 0xFE
Left-most byte shifted: 0x5A << 8 = 0x5A00
Adding in the right most byte: 0x5A00 + 0x00FE = 0x5AFE
```

Similarly, storing the integer requires shifting the leftmost byte to the right to preserve the values. I leave this for you to explore as an exercise. Listing 4-7 shows two methods for reading and storing integer values in a byte buffer. Take some time to read through these should you need to read or store integers in a byte stream.

Listing 4-7. Reading and Storing Integers in a Byte Array

```
/**
 * get_lcb_len - Retrieves the length of a length coded binary value
 *
 * This reads the first byte from the offset into the buffer and returns
 * the number of bytes (size) that the integer consumes. It is used in
 * conjunction with read_int() to read length coded binary integers
 * from the buffer.
 *
 * Returns integer - number of bytes integer consumes
*/
int get_lcb_len(byte buffer[], int offset) {
  int read_len = buffer[offset];
  if (read_len > 250) {
    // read type:
    byte type = buffer[offset+1];
    if (type == 0xfc)
      read_len = 2;
    else if (type == 0xfd)
      read_len = 3;
    else if (type == 0xfe)
      read_len = 8;
  }
  return 1;
}
/**
 * read_int - Retrieve an integer from the buffer in size bytes.
 *
 * This reads an integer from the buffer at offset position indicated for
 * the number of bytes specified (size).
 *
 * buffer[in]      byte stream in memory
 * offset[in]      offset from start of buffer
 * size[in]        number of bytes to use to store the integer
```

```
 *
 * Returns integer - integer from the buffer
*/
int read_int(byte buffer[], int offset, int size) {
  int value = 0;
  int new_size = 0;
  if (size == 0)
    new_size = get_lcb_len(offset);
  if (size == 1)
    return buffer[offset];
  new_size = size;
  int shifter = (new_size - 1) * 8;
  for (int i = new_size; i > 0; i--) {
    value += (byte)(buffer[i-1] << shifter);
    shifter -= 8;
  }
  return value;
}


/**
 * store_int - Store an integer value into a byte array of size bytes.
 *
 * This writes an integer into the buffer at the current position of the
 * buffer. It will transform an integer of size to a length coded binary
 * form where 1-3 bytes are used to store the value (set by size).
 *
 * buff[in]        pointer to location in internal buffer where the
 *                 integer will be stored
 * value[in]       integer value to be stored
 * size[in]        number of bytes to use to store the integer
*/
void store_int(byte *buff, long value, int size) {
  memset(buff, 0, size);
  if (value < 0xff)
    buff[0] = (byte)value;
  else if (value < 0xffff) {
    buff[0] = (byte)value;
    buff[1] = (byte)(value >> 8);
  } else if (value < 0xffffff) {
    buff[0] = (byte)value;
    buff[1] = (byte)(value >> 8);
    buff[2] = (byte)(value >> 16);
  } else if (value < 0xffffff) {
    buff[0] = (byte)value;
    buff[1] = (byte)(value >> 8);
    buff[2] = (byte)(value >> 16);
    buff[3] = (byte)(value >> 24);
  }
}
```

■ **Note** This code is meant to be an example of how to encode and decode. If you want to use this in your own solution, you may need to modify it to fit your code.

Notice one method uses a loop to iterate over the bytes while the other uses the more common approach of using a conditional statement. I include both, but you can use whichever is easiest to understand. Notice that both methods require the size of the integer. This is necessary because the integer could be 1, 2, or 4 bytes in length.

Another concept of data type usage concerns values that never change. In this case, you can declare the variable as a constant, which tells the compiler to substitute the variable with the constant value, thus saving a small bit of memory. Do this with larger data types and you could save a lot of memory. Consider the following code snippet. The first line of code uses 2 bytes, while the second forces the compiler to substitute 10 everywhere the variable appears.

```
int multiplier = 10;
const int multiplier = 10;
```

There are quite a number of tricks and techniques for working with data types—so much so that entire multivolume books have been written to explore the nuances and pros and cons of every technique possible. If you are using an Arduino in your IOT solution and want to ensure you are programming it with as much care and efficiency as possible, see Dr. Simon Monk's book *Programming Arduino Next Steps* (McGraw Hill, 2014).

If you are programming your low-cost computer board with Python, many of these techniques are the same or have similar application. An excellent book on efficient Python programming on the Raspberry Pi is Wolfram Donat's book *Learn Raspberry Pi Programming with Python* (Apress, 2014). If you want to go beyond the basics of Python programming, check out J. Burton Browning and Marty Alchin's book *Pro Python* (Apress, 2014).

Database Considerations

Fortunately, you are not likely to encounter the same memory limitations when storing data in a database that you would with a microcontroller or similar device with small memory sizes. This is because databases are good at storing data in specific formats and in some cases can optimize the storage used (especially for text). Perhaps more importantly, database servers have a lot more storage space than a microcontroller.

Thus, there really isn't an issue storing data in its proper type for a database with some minor exceptions. That is, there are some possible border cases where complex data types may not be available on the database server, but there are certainly enough primitives that you can save whatever type you need.

■ **Note** MySQL supports many data types. See the section entitled "Data Types" in the MySQL online reference manual for more details (`http://dev.mysql.com/doc/refman/5.7/en/`).

Furthermore, the issue of dealing with scale is not an issue for storing data. This is because, once again, the database is really good at that. Thus, rounding or limiting decimals for floating-point numbers is more about presentation than anything else. However, if you need to do something like this, you can in the database server.

For example, MySQL provides additional controls for floating-point data types. That is, you can set the number of whole digits as well as the number of decimals. You can do this with the `float(m,d)` syntax where the `m` is the maximum number of digits and `d` is the number of decimals. Use this syntax when you need to limit the size or display of floating-point numbers in MySQL.

Adding Derived or Calculated Data

Another common annotation is adding additional data to the row that is a derivation or calculation of the original value. This includes values that are converted to change a unit of measure (Fahrenheit to Celsius), enumeration based on ranges or transposed in some manner (scale, precision), or calculated columns (arithmetic operations) where the result has meaning.

For example, there may be sensor data generated for a group of events or sensors that are used together with other sensors to provide a result or perhaps situations where you want to scale, divide, convert the data to a new unit of measure, or otherwise modify a value. In these cases, we add the new derived or calculated data to the row. Let's consider you have a sensor that measures distance in inches but you need the data to be millimeters.[6] You could convert the data to inches easily with the following formula:[7]

```
value_millimeters = value_inches * 25.4;
```

Recall we never want to discard the original value, so we would save both the value in inches as well as the derived value. Thus, we would write the data as follows in a now-familiar Python script excerpt:

```
strData = "Distance milimeters: {0:.2f}, Distance inches: {1:.2f} Datetime: {2}\n"
file_log = open("data_log_python.txt", 'a')
dist_inches = read_sensor(1);
dist_mm = dist_inches * 25.4;
file_log.write(strData.format(dist_mm, dist_inches))
file_log.close()
```

Another common derived column is the use of a column to store the amount that a value has changed since the last reading. While it is possible to calculate this from saved data, adding an additional column to store the value can make reading and understanding the data easier. To do this, we would have to save the old value and calculate the change by subtracting it from the new value. A positive result is the amount the value increased; a negative value is the amount the value decreased.

Let's take a closer look at an example of common derived and calculated columns.

Code Implementation

Derived and calculated data (columns) will be implemented in a specific manner. That is, there will be a precise formula or translation that needs to be performed. In the following example, we will see three samples of derived and calculated columns that I have encountered in my own and others' IOT solutions.

The first is a derivation for calibration. Sometimes sensors need to be calibrated and the result of the calibration will determine a value (sometimes a formula if the difference is not linear) that must be added or subtracted to get a more accurate value.

The second is an example of a simple calculation where we have a sensor that measures weight on a plate or platter for a number of objects. In the example, the number of objects is fixed, but in most cases this will also be a variable. Savvy IOT hobbyists and enthusiasts will note calculating the average weight of irregular objects may not be interesting, so we assume the objects for this example are all similar in size and composition.

---

[6]Or more likely inches to centimeters.
[7]I avoid the quaint but over used temperature example. You're welcome.

The third is another example of a derived value where we want to store the difference of the value read from the last value read. This represents a host of annotations involving averages, running totals, and so on, involving numeric data.

While all of these are technically changing or adding new data, we keep the original values to ensure we can recover from any changes in the derivation or calculations. Also, we store these values to make reading and processing the data easier.

Listing 4-8 shows an excerpt from an Arduino sketch that simulates and implements reading of three sensors: a blood oxygen sensor that requires calibration,[8] a weight sensor measuring weight of several objects, and a voltage sensor that we use to keep the delta since the last value read.

Listing 4-8. Derived and Calculated Annotations (Arduino)

```
/**
  Example of derived or calculated columns annotation.

  This project demonstrates how to save data to a
  microSD card as a log file with additional column annotation.
*/
#include <SPI.h>
#include <SD.h>
#include <String.h>

// Pin assignment for Arduino Ethernet shield
//#define SD_PIN 4
// Pin assignment for Sparkfun microSD shield
#define SD_PIN 8
// Pin assignment for Adafruit Data Logging shield
//#define SD_PIN 10

File log_file;   // file handle
String strData;  // storage for string
float blood_oxygen;

// Simulated sensor reading method
float read_sensor(int sensor_num) {
  if (sensor_num == 1) {
    return 90.125 + random(20)/10.00;       // sensor #1
  } else if (sensor_num == 2) {
    return 94.512313 + random(100)/100.00;  // sensor #2
  } else if (sensor_num == 3) {
    return 45.6675 + random(100)/100.00;    // sensor #3
```

---

[8]In this case, I use a simple calibration of shifting the value by a fixed delta. Calibrations of this nature are seldom. Calibrations may be based on a linear scale (the inaccuracy gets worse or less as values increase) or could require a more complex formula to correct the values.

```
  } else {
    if (random(25) >= 5) {
      return (float)random(14)/10.00;
    } else {
      return -(float)random(14)/10.00;
    }
  }
}

void setup() {
  Serial.begin(115200);
  while (!Serial);
  Serial.print("Initializing SD card...");
  if (!SD.begin(SD_PIN)) {
    Serial.println("ERROR!");
    return;
  }
  Serial.println("ready.");

  if (SD.remove("data_log.txt")) {
    Serial.println("file removed");
  }

  // initiate random seed
  randomSeed(analogRead(0));
}

float oldVal = 67.123;
float newVal = 0.00;
float weight = 0.00;

void loop() {
  delay(2000);
  log_file = SD.open("data_log.txt", FILE_WRITE);
  if (log_file) {
    strData = String("Blood oxygen: ");
    blood_oxygen = read_sensor(2);
    strData += blood_oxygen;
    strData += ", Calibrated: ";

    // calculated column adjusting for calibration
    strData += String(blood_oxygen + 0.785, 2);
    strData += ", Weight: ";
    weight = read_sensor(3);
    strData += weight;

    // calculated column for number of objects
    strData += ", Avg weight: ";
    strData += String((weight/4.0), 4);

    // Calculating change since last read
    strData += ", Volts: ";
    newVal = oldVal+read_sensor(4);
```

125

```
    strData += String(newVal, 2);
    strData += ", Delta: ";
    strData += String(newVal-oldVal,3);
    oldVal = newVal;

    log_file.println(strData);
    log_file.close();
  } else {
    Serial.println("Cannot open file for reading.");
  }
}
```

■ **Note**   The following code is not representative of an actual solution; rather, it is designed to show the concepts of generating derived and calculated columns.

Notice once again we add the new data as columns by separating them by commas. The following shows a sample set of rows from this code:

```
Blood oxygen: 94.88, Calibrated: 95.67, Weight: 46.07, Avg weight: 11.5169, Volts: 68.42,
Delta: 1.300
Blood oxygen: 95.23, Calibrated: 96.02, Weight: 46.56, Avg weight: 11.6394, Volts: 68.52,
Delta: 0.100
Blood oxygen: 94.97, Calibrated: 95.76, Weight: 46.42, Avg weight: 11.6044, Volts: 68.62,
Delta: 0.100
Blood oxygen: 95.46, Calibrated: 96.25, Weight: 46.49, Avg weight: 11.6219, Volts: 69.62,
Delta: 1.000
Blood oxygen: 94.96, Calibrated: 95.75, Weight: 46.18, Avg weight: 11.5444, Volts: 70.12,
Delta: 0.500
Blood oxygen: 94.62, Calibrated: 95.41, Weight: 46.11, Avg weight: 11.5269, Volts: 70.52,
Delta: 0.400
```

Notice here we see the blood oxygen sensor, and its calibrated value is stored along with the weight and average weight of four objects and then volts and the change since the last value read is stored. Let's now see how the database server can make each of these annotations easier.

Database Considerations

By now you should be thinking the database server is a powerful tool, and it is. In fact, it is great at doing the derivations and calculations like those shown earlier. Moreover, depending on the calculation needed, you have several options for how to implement the derivation or calculation. This section discusses three common alternatives. There are other alternatives,[9] but these are among the most common methods.

You could add a trigger, which is a special procedure that is executed (triggered) when data is added; to add the new data in a special column, you could put the calculations in the SELECT statement (the method for retrieving rows) so that the new data is generated on the fly; or you could put the calculations in the INSERT statement (the method for saving data). Let's look at each of these starting with why you may choose one over the other.

---

[9]Such as stored events, stored procedures, and so on.

The first thing you should consider is whether to generate the new data on the fly or as a column stored in the table. Generating the new value on the fly will mean less storage used but may come as a premium when retrieving the rows. That is, it may take longer to retrieve a large number of rows since the calculation is deferred and done all at once. Thus, saving the new data in the table means less time spent retrieving the rows but does require a bit more storage space. Fortunately, storage space isn't usually the issue. Note that the calculation specified in the SELECT statement is still executed on the database server albeit once for each row as it is retrieved.

As to whether to generate the data in the database with a trigger or as part of the INSERT statement, we should consider where we want the calculation to be executed. If placed in a trigger, the calculation is performed on the database server. But if placed in the INSERT statement, the calculation is performed on the client (the one sending the data). Thus, if you have complex calculations or limited processing power, you may want to choose the trigger option.

Before we look at the database examples, the following shows the table needed for executing the examples. I include this here so that you can test these examples yourself and so that you can see the only new column added is the calibrated column, which is used by the trigger example. Listing 4-9 shows the test table.

Listing 4-9. Table Schema for Derived and Calculated Columns Example

```
CREATE TABLE `derived_annotation_test` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `blood_oxygen` float NOT NULL,
  `blood_oxygen_corrected` float DEFAULT NULL,
  `weight` float DEFAULT NULL,
  `volts` float DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1
```

Notice we add a column for the calibration of the blood oxygen example but not for the other two values. Those we will generate on the fly. Now let's see how each of the three options can be implemented in code.

Derived and Calculated Columns Using a Trigger

A trigger is a good way to program your database to populate a column. In this case, we want to write a trigger that executes each time a row is added to the table and, when it does, calculate the calibrated value using a simple formula. A trigger can be executed (or *triggered*) in several ways such as before or after an insert or delete. A trigger is associated with a specific table, but a table may have more than one trigger. We will discuss triggers in more detail in Chapter 5, but for now observe the following:

```
CREATE TRIGGER calibrate_blood_oyxgen BEFORE INSERT ON derived_annotation_test
FOR EACH ROW SET NEW.blood_oxygen_corrected = NEW.blood_oxygen + 0.785;
```

What we see here is a trigger set to execute before an insert for the sample table named derived_annotation_test. Notice the second line reads quite clearly that a new value will be set for the blood_oxygen_corrected column with the value of the blood_oxygen value plus 0.785.

Now let's see what happens when we insert a row. The following inserts a row in the table. Notice I specified a list of columns followed by a list of values. If you do not specify the column list, you must supply values for all of the columns in the table.

```
INSERT INTO derived_annotation_test (blood_oxygen, weight, volts) VALUES (94.88, 46.07,
68.42);
```

Now let's see what the data looks like. Recall to get data from the table, we use the SELECT command as follows:

```
mysql> SELECT * FROM  derived_annotation_test;
+----+-------------+-----------------------+--------+-------+
| id | blood_oxygen | blood_oxygen_corrected | weight | volts |
+----+-------------+-----------------------+--------+-------+
|  5 |       94.88 |                95.665 |  46.07 | 68.42 |
+----+-------------+-----------------------+--------+-------+
1 row in set (0.00 sec)
```

Here we see the corrected value has been saved in the blood_oxygen_corrected column. Best of all, we didn't have to do anything when we added the data. Indeed, the beauty of a trigger is you set it once and it works for all data inserted.

■ **Note** Some of the Python code examples use a special library for connecting to MySQL called a *database connector*. In this case, it is the Connector/Python connector library from Oracle (http://dev.mysql.com/downloads/connector/python/). You will see this connector in much more detail in Chapters 6 and 8.

Derived and Calculated Columns Using a SELECT Statement

Recall we can also generate derived or calculated columns on the fly as the data is read. More specifically, we include the operation as part of the SELECT statement. Let's insert some data to see how this is done. In this example, we will show how to calculate the difference since the last value read for the volts column. Recall from the sample table, we do not have a column to store the delta. The following shows an example INSERT that inserts multiple rows. This is also called a *bulk insert*.

```
INSERT INTO derived_annotation_test (blood_oxygen, weight, volts)
VALUES (94.88, 46.07, 68.42), (95.23, 46.56, 68.52), (94.97, 46.42, 68.62);
```

Notice we simply supply a comma-separated list of values, each representing a new row of data. In this case, we use the same data as shown in the earlier Arduino sketch. Once the data is inserted, we can view it as follows:

```
mysql> SELECT * FROM derived_annotation_test;
+----+-------------+-----------------------+--------+-------+
| id | blood_oxygen | blood_oxygen_corrected | weight | volts |
+----+-------------+-----------------------+--------+-------+
|  5 |       94.88 |                95.665 |  46.07 | 68.42 |
|  7 |       94.88 |                95.665 |  46.07 | 68.42 |
|  8 |       95.23 |                96.015 |  46.56 | 68.52 |
|  9 |       94.97 |                95.755 |  46.42 | 68.62 |
+----+-------------+-----------------------+--------+-------+
4 rows in set (0.00 sec)
```

To calculate the delta, we write a Python script to connect to the database and retrieve the rows from the table. We begin by saving the value of the first value read and then compare it on subsequent rows, calculating the change since the last value was read. Listing 4-10 shows a Python script that uses the MySQL Connector/Python library to read data from the server.

Listing 4-10. Calculated Columns Using SELECT (Python)

```
import mysql.connector;

cnx = mysql.connector.connect(user="root", password="SECRET")
cur = cnx.cursor()
cur.execute("SELECT * FROM test.derived_annotation_test")
old_value = 0;
for row in cur.fetchall():
  if old_value > 0:
    print "{0}, {1}, {2}, {3}, {4}".format(row[1], row[2], row[3], row[4], row[4] - old_
    value)
  else:
    print "{0}, {1}, {2}, {3}".format(row[1], row[2], row[3], row[4])
  old_value = row[4]
```

While there may be more efficient ways to read the column values, I wrote the code to access each column in the row to show you how the result is returned as a list (like an array). I simply access the column number, which starts at 0, to get the value for the column. Note the value for the `volts` column is the fifth column in the table and therefore is referenced with [4]. The following shows the output of running this script:

```
$ python ./derived_select_example.py
94.88, 95.665, 46.07, 68.42,
94.88, 95.665, 46.07, 68.42, 0.0
95.23, 96.015, 46.56, 68.52, 0.1
94.97, 95.755, 46.42, 68.62, 0.1
```

Here we see the code prints out the data for the `blood_oxygen`, `blood_oxygen_corrected`, `weight`, and `volts`. There is no value for the first row since we don't have a previous value to use to calculate the delta. The remaining rows show the delta or change since the last value as a fifth value.

Derived and Calculated Columns Using an INSERT Statement

Now let's look at the INSERT method. We will use the volts derived data from earlier. In this case, we have stored the old value at some point in the code so that when we save the new value, we include the calculation there. The following code is an example of how to form the INSERT statement to include calculations. While the formula is simple, the example shows how it is possible to populate derived or calculated columns when the data is inserted.

Recall the calculation is performed on the client as opposed to the trigger example, which is executed on the server. I leave for an exercise converting this example to a trigger. Before we see the SQL code, let's add the calculated column to the table. There is a powerful command named ALTER TABLE, which you can use to add or remove columns, and much more. We use it to add the new column as follows:

```
ALTER TABLE derived_annotation_test ADD COLUMN avg_weight float AFTER weight;
```

Now we add the new data. Recall from the Arduino example, we want to simply take the weight value and divide it by 4 to get the average weight for each item the sensor has measured as weight. The following is another bulk insert statement. Notice we simply include the formula in the space for the average weight column.

```
INSERT INTO derived_annotation_test (blood_oxygen, weight, avg_weight, volts)
VALUES (95.46, 46.49, (46.49/4.00), 69.62), (94.96, 46.18, (46.18/4.00), 70.12),
(94.62, 46.11, (46.11/4.00), 70.52);
```

A quick check of the data in the table confirms the rows were inserted with the correct values.

```
mysql> SELECT * FROM derived_annotation_test;
+----+--------------+------------------------+--------+------------+-------+
| id | blood_oxygen | blood_oxygen_corrected | weight | avg_weight | volts |
+----+--------------+------------------------+--------+------------+-------+
|  5 |        94.88 |                 95.665 |  46.07 |       NULL | 68.42 |
|  7 |        94.88 |                 95.665 |  46.07 |       NULL | 68.42 |
|  8 |        95.23 |                 96.015 |  46.56 |       NULL | 68.52 |
|  9 |        94.97 |                 95.755 |  46.42 |       NULL | 68.62 |
| 10 |        95.46 |                 96.245 |  46.49 |    11.6225 | 69.62 |
| 11 |        94.96 |                 95.745 |  46.18 |     11.545 | 70.12 |
| 12 |        94.62 |                 95.405 |  46.11 |    11.5275 | 70.52 |
+----+--------------+------------------------+--------+------------+-------+
7 rows in set (0.00 sec)
```

Here we see only the last three rows have the average weight column. Recall we added the new column to a table that already contained data. Unless there was a trigger added, the values for the new column for the existing rows will remain empty (technically NULL).

Data Interpretations

Sometimes data is generated in a form that is not usable or requires some translation or interpretation to be usable. We have already seen an example of data interpretation. Recall the plant-monitoring example from Chapter 1 where we had a sensor that, depending on the value generated, could indicate one of several states. Thus, we can interpret the data values to be one of these states and therefore store the state. In this case, we are creating a value derived from a range of values rather than calculating a new value.

Let's see a similar example, but this time we will see data generated from a different type of sensor. In this case, the sensor being simulated is a liquid-level sensor such as sold by Sparkfun (http://sparkfun.com/products/10221). This is a sensor that measures resistance that can be used to determine the distance from the top of the sensor to the surface of the liquid. More specifically, the output is inversely proportional to the level of the liquid. The lower the liquid, the higher resistance measured. The lower the liquid, the less resistance measured.

Now, let's suppose we are using this sensor in a pond-monitoring solution where we want to measure the level of the water inside the filter reservoir. Furthermore, we know from experience and observation that depending on the level of the water, the state of the pond can change as follows:

- If the water level is low at 6" or lower than a fixed point, the filters may need cleaning.

- If it water level is between 3-6", the pond is low on water.

- If the water level is between 1-3", the pond is in a normal state.

- If the water level is above 1", there is too much water in the pond and water is flowing over the filter elements.

Thus, we need to check the value for resistance in specific ranges and store an enumerated value for the range. We will name the range (CLEAN, LOW, NORMAL, HIGH). Now let's see how we can store a row of data using a Python script. Remember, we always want to store the original value so that if the enumerations need adjusting (the ranges change), we can alter the existing data without rendering it unusable.

Code Implementation

In this example, we use Python code to determine which enumeration to use. The following code excerpt is taken from another Python script that uses the Connector/Python library. Don't worry too much about the mechanics of the library; rather, notice how I use Python code to effect the enumeration.

The values read from the sensor are in ohms in the range (300 to 1500 +/-10%). So how do we know what ranges to use? Do we install the sensor and take some measurements? We could do that. In fact, we could use a tall container and fill it with water slowly while observing the values from the sensor. This will work and is a good way to test the sensor, but there is another, faster way to get started.

Most manufacturers provide what is called a *data sheet* that describes how the device performs. In this case, the manufacturer provides a graph that indicates what values to expect for certain levels of liquid (http://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/eTape%20Datasheet%2012110215TC-8_040213.pdf). From this data, we can determine the following ranges correspond to the water levels observed. Table 4-2 shows the complete data.

Table 4-2. *Liquid-Level Enumerated Values for Pond Monitoring*

Depth from Top of Sensor in Inches	Value Range	Result
0 < 1	1500 and higher	HIGH
1 < n < 3	1150 to 1500	NORMAL
3 < n < 6	1150 to 700	LOW
6 < n	700 and lower	CLEAN

Notice how some of the ranges overlap. Only use or experimentation will determine the actual values, but these should provide a good start. Now let's see the Python code (see Listing 4-11). Here we are constructing an INSERT statement that includes the derived data from code preceding the execution of the INSERT statement.

Listing 4-11. Derived Values Example (Python)

```python
import mysql.connector;
from random import random, randint

def read_sensor():
  return 500 + randint(0,1500)

strInsert = "INSERT INTO pond VALUES (null, {0}, '{1}')"
cnx = mysql.connector.connect(user="root", password="SECRET", database="test")
cur = cnx.cursor()
# Calculate enumerated value for sensor.
water_level = read_sensor()
if water_level > 1500:
  state = 'CLEAN'
elif water_level > 1150:
  state = 'LOW'
elif water_level > 700:
  state = 'NORMAL'
else:
  state = 'HIGH'
```

```
cur.execute(strInsert.format(water_level, state))
cnx.commit()
cur.close()
cnx.close()
```

Take a moment to read through the code. Don't worry too much about the connector library. Focus instead on the code used to set the value for the state. It is rather straightforward, and the value determined is used in the INSERT statement via parameter substitution.

To test the code, we need to create a test table as follows. Notice I add the two columns we will need along with an auto-increment column for ease of identifying the rows.

```
CREATE TABLE `pond` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `water_level` int NOT NULL,
  `state` char(12) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Next, we can run the script several times to create several rows. Savvy Python developers will see I simulated the sensor read code with a randomly generated value. Clearly, this will create quite a number of rows with wildly varying values. That is, in an actual pond, the water level would not fluctuate so radically from one moment to another. However, the code is sufficient to test and demonstrate the technique of enumerated values. Once several rows have been inserted, we can see the results as follows:

```
mysql> SELECT * FROM pond;
+----+-------------+--------+
| id | water_level | state  |
+----+-------------+--------+
|  8 |        1054 | NORMAL |
| 10 |        1117 | NORMAL |
| 11 |        1278 | LOW    |
| 12 |        1316 | LOW    |
| 13 |        1451 | LOW    |
| 14 |        1688 | CLEAN  |
+----+-------------+--------+
6 rows in set (0.00 sec)
```

Take a moment to ensure the values match the enumerated value based on the data in Table 4-2. I encourage you to run the script a number of times until you see an example for each state. Hint: you may want to alter the read_sensor() method to return a random number for each range instead of a range from 0 to 1500.

Database Considerations

There are several ways to handle data interpretation in the database. I will discuss two such methods: using enumerated values where we determine the state when we save the data (still requires a bit of code) and using an advanced technique to determine the state automatically when the data is saved or updated. Yes, another trigger!

By now we know that triggers can be powerful, and in this case we can include a similar set of code to determine the state value from the sensor. In fact, it resembles the Python example. Let's look at the code inside the trigger to focus on how the interpretation is accomplished. The following shows the trigger code:

```
DELIMITER //
CREATE TRIGGER set_pond_state BEFORE INSERT ON pond FOR EACH ROW
BEGIN
  IF NEW.water_level > 1500 THEN
    SET NEW.state = 'CLEAN';
  ELSEIF NEW.water_level > 1150 THEN
    SET NEW.state = 'LOW';
  ELSEIF NEW.water_level > 700 THEN
    SET NEW.state = 'NORMAL';
  ELSE
    SET NEW.state = 'HIGH';
  END IF;
END //
DELIMITER ;
```

Now let's test the trigger using a bulk insert. Here we will insert four new rows with values chosen specifically to result in one of the four states.

```
INSERT INTO pond VALUES (null, 1501, null), (null, 1151, null), (null, 701, null), (null,
600, null);
```

There is an interesting system variable in MySQL. It is named `last_insert_id` and stores the last value used for the auto-increment field. If you exercised the Python example, you will have already created the table and inserted several rows. Rather than display (return) all of the rows, we can use the `last_insert_id` to find the last value of the auto increment mechanism. In this case, the value is the last value used for the last insert, but since this was a bulk insert, the value is the first value in the bulk (in this case 15). Try it yourself.

```
mysql> select @@last_insert_id;
+------------------+
| @@last_insert_id |
+------------------+
|               15 |
+------------------+
1 row in set (0.00 sec)
```

The following shows the rows inserted:

```
mysql> SELECT * FROM pond WHERE id >= 15;
+----+-------------+--------+
| id | water_level | state  |
+----+-------------+--------+
| 15 |        1501 | CLEAN  |
| 16 |        1151 | LOW    |
| 17 |         701 | NORMAL |
| 18 |         600 | HIGH   |
+----+-------------+--------+
4 rows in set (0.01 sec)
```

Notice we see the four rows we inserted, and the correct enumerated values have been chosen. I should also note at this point that there is a data type called enum, which you can use to store the strings for the values in the table itself. We will see this option in action in Chapter 5; however, if you are familiar with using enum, I encourage you to change the table and trigger shown previously to use the enum instead of a character string.

Now that you have seen several examples of annotation, let's discuss a more complex application of aggregating data from multiple sensors or nodes as well as the type of aggregation operations you may want to implement in your IOT solutions.

Aggregation

Aggregating data for your IOT solution can be a more complex operation than annotating the data. Aggregation can take several forms. The easiest form to implement is aggregating data from multiple sensors. Similar in concept but quite a bit different in implementation is aggregating data from multiple nodes (other data collectors). Finally, the more complex form of aggregation is performing operations on sets of data such as statistical or counting functions.

In this section, we explore each of these forms of aggregation from a high level. Since the implementation of each can be complex, I present general overviews rather than specific code solutions. That said, we will see some of these aggregation forms implemented in Chapter 8. Thus, the following sections discuss strategies, practices, and general descriptions of each form.

Data from Multiple Sensors

It is normal for IOT solutions to collect data from multiple sources or sensors. Sometimes the sensors are used to observe different things (events, objects, and so on), but most times several sources or sensors are used to observe a single thing or subject. That is, you may have a data collection node in the form of an Arduino board reading data from several sensors monitoring something. Or perhaps you may have a single Arduino reading many sensors that monitor several subjects.

The question then is how do you store the data? Should you store the sensor values as a set like you've seen in the earlier examples in this chapter, or should you store the values separately? There are two approaches: storing data based on sensor timing and storing data based on specific time periods.

Sensor-Driven Data

Data that is generated based on the timing of sensor reads means the data is stored at intervals dictated by the availability of the data. Further, the data is used (displayed, tabulated or mined for information, and so on) independently when the data is collected. That is, the time when the data is used has no meaning for the data.

While it is natural to store a set of sensor values that correspond to a single subject, if the sensors produce values at different intervals, you may have to consider storing the values one at a time rather than waiting for all sensors to report. In fact, it is not unusual for different sensors to be read at different intervals. This may be because of the nature of what you are observing and where the timing of the observation can help produce knowledge.

For example, consider the plant-monitoring solution. You may want to read temperature once every hour (or even more frequently) because outdoor temperature can change quickly in some areas in an hour, or you may want to read temperature every few hours for indoor, controlled environments. However, for soil moisture, you may want to read the values twice a day since soil moisture may not change as quickly, especially for controlled climates.

Let's say you decide you want to read temperature values every hour and soil moisture values every six hours. Given those intervals, what do you do with the six temperature values? You would have six temperature values for each soil moisture value. Do you average the temperature values or throw away five of them?

Clearly, discarding five sensor readings is a potential loss of information. In this case, you could lose data about when the temperature changed. For example, if the temperature changed at the first hour by 4 degrees (not unusual in my area) but only 1 degree over the next five hours, saving the last value obscures when the temperature changed and more importantly loses the time event of the rapid change of temperature. Even averaging the values will lose the data and obscure knowledge. The loss of knowledge may not be obvious and requires a bit of thought. Table 4-3 shows an example of the type of data we could collect.

Table 4-3. *Sensor Data Frequency and Loss of Knowledge*
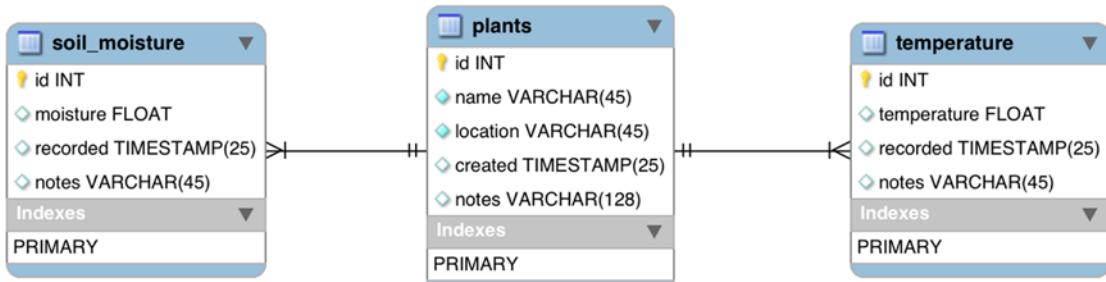
Hour	Temperature	Soil Moisture
1	24.5	
2	24.7	
3	24.9	
4	25.2	
5	25.4	
6	25.6	426
7	25.8	
8	27.9	
9	30.1	
10	29.3	
11	28.9	
12	28.6	410

Notice here we see values for the temperature (in Celsius) but only one for the soil moisture each six-hour period. If we stored the temperature read only when the soil moisture is read, we would see a large change in value and would not know when the temperature changed—only that it changed since the last temperature read six hours prior.

For example, notice the temperature and soil moisture for hour 6. Here we see we stored the values (25.6, 426) respectfully. Notice the values for hour 12. Here we stored the values (28.9, 410). While there wasn't much change for the soil moisture, we see a change of temperature of (28.9 – 25.6 = 3.0). However, we have lost the moment when the temperature changed the most between hours 7 and 10 and even the fact that the temperature was highest at hour 9.

Conversely, if we averaged the temperature values read, we would save data at hour 6 of (25.05, 426) and hour 12 of (28.43, 410). While we have factored in the values over time, we haven't gained much more information. Yes, we still detect the trend of the temperature rising between the intervals, but the hour where the temperature was highest is still lost. You can also say we've lost the knowledge of the rate of change as well as even accuracy since we are storing values for temperature which are not accurate for the time the value was read.

When you encounter situations where storing the sensor data as a single row will obscure knowledge, you are going to need to divide your data over two tables instead of one. Figure 4-2 shows an example of a solution where we can save sensor data at different rates but still associate it with a single thing.

Figure 4-2. *Collecting data from multiple sensors at different rates*

Notice here we have three tables. The base table or the table with the core information is named `plants`. We see a separate table to store soil moisture data and another to store temperature. The tables share a common column, `id`, which is used to uniquely identify the plant for which the sensor data was read. That is, we can retrieve all the temperature or soil moisture values for a given plant (the column `id` is a foreign key for the `temperature` and `soil_moisture` tables). Notice we also store a timestamp for each row so we know when the sensor data was stored and therefore can plot data over time or determine rates of change.

■ **Note**   The drawing in Figure 4-2 was taken from the visual database editor in MySQL Workbench (http://dev.mysql.com/downloads/workbench/).

By making three tables instead of one we have preserved the potential to gain knowledge from the data, which is being read at different intervals. Indeed, this example shows how powerful the database can be. Sure, you can implement this solution in a file-based solution, but the database makes it far easier to see and even easier to set up.

Interval Driven Data

There are solutions where it is more important to store data on a fixed or ad hoc interval. In these solutions, the focus is on gathering the data for use at a specific time. Thus, it is likely the solution will not store data unless or until the data is requested. Some solutions make the data available dynamically but generally do not store it until the user or some other mechanism initiates the request.

For example, consider a weather station. If you were building a weather station yourself, you would probably incorporate a display element to show the current values of weather observations such as temperature, barometric pressure, and so on, in real time. In this case, the data from the sensors is read frequently, and the display element (for example, gauge) is updated frequently. However, the data for these values is not likely to be stored.

Rather, for a weather station, the data saved is typically a set of observations at a specific time period. You may see weather stations that store data once per hour or even several times per hour. This is because the observations have a common data element—time of collection. Thus, interval driven data may require you to build your solution to store a host of data at specific intervals and not at the interval of a specific sensor or event.

For these solutions, we typically store the information as a single entry (row). However, there is nothing that says we have to do so. We still may want to store the data in separate tables (files) if there is a need to do so, but there isn't normally an issue related to the data or knowledge gained. Again, this is because the data is saved as specific intervals across the sensors rather than being driven by when sensor data is available.

Data from Multiple Nodes

Closely related to data from multiple sensors is data collected from several nodes. Rather than reading sensors directly, the node is getting data from other devices. For example, you may have an Arduino connected to several modules wirelessly using XBee modules. Since the XBee modules can only send data, you need the Arduino node to gather the data and store it—either in a file locally or in a database.

The same considerations apply as we discovered with data from multiple sensors. It is most likely each XBee module is sending data at regular intervals and may include multiple pieces of data (multiple sensor data). Thus, how to save the data has the same implications—storing the data together as a single row or storing the data separately.

So far, nothing is different in principle from the last form. However, collecting data from multiple nodes has another capability that can be exploited—saving data in bulk. Recall from a previous example, the MySQL INSERT SQL statement permits storing data for several rows in a single command. By collecting data from multiple nodes, we can build such statements easily.

Listing 4-12 shows an excerpt from an Arduino sketch that gets data from multiple XBee modules and saves the data in a MySQL database.[10] In this example, I keep things rather simple where we are reading sensor data generated from multiple data nodes represented as an XBee module with a temperature sensor. The code is designed to bulk insert data into the database. I have set the number of rows included at three, but you can easily expand this code to as many rows as you have memory to store the data.

So how do we know which XBee was read? We save the address of the module—all XBee modules have a unique address, and thus we know the origin of the temperature data. We also see a case of performing calculations on the data node where we store the temperature in its raw sensor form as well as in Celsius and Fahrenheit. I should also note that with multiple XBee modules connected, it is possible to gather data from the same module twice depending on the timing of when the XBee modules send data.

Listing 4-12. Getting Data from Multiple Nodes (Arduino)

```
String get_data(ZBRxIoSampleResponse *ioSample) {
  // Received data from address of data node
  int address = (ioSample->getRemoteAddress64().getMsb() << 8) +
                 ioSample->getRemoteAddress64().getLsb();

  // Get and calculate the temperature in C and F
  float temp_raw = ioSample->getAnalog(3);
  float temp_c = ((temp_raw * 1200.0 / 1024.0) - 500.0) / 10.0;
  float temp_f = ((temp_c * 9.0)/5.0) + 32.0;
  String strRow = String("(");
  strRow += String(address);
  strRow += ",";
  strRow += String(temp_raw);
  strRow += ",";
  strRow += String(temp_c);
  strRow += ",";
  strRow += String(temp_f);
  strRow += ")";

  return strRow;
}

...
```

---

[10]Yes, you can do that with an Arduino!

137

```
void loop() {
  String row[3];
  int i;

  i = 0;
  //attempt to read a packet
  xbee.readPacket();
  if (xbee.getResponse().isAvailable()) {
    // XBee module is communicating, check for IO packet
    if (xbee.getResponse().getApiId() == ZB_IO_SAMPLE_RESPONSE) {
      // Get the packet
      xbee.getResponse().getZBRxIoSampleResponse(ioSample);
      row[i] = get_data(&ioSample);
      i++;
    }
    else {
      Serial.print("Expected I/O Sample, but got ");
      Serial.print(xbee.getResponse().getApiId(), HEX);
    }
  } else if (xbee.getResponse().isError()) {
    Serial.print("Error reading packet.  Error code: ");
    Serial.println(xbee.getResponse().getErrorCode());
  }
  // Store the data once 3 entries are retrieved.
  if (i == 3) {
    i = 0;
    String strINSERT = String("INSERT INTO test.room_temperatures VALUES ");
    strINSERT += row[0];
    strINSERT += ",";
    strINSERT += row[1];
    strINSERT += ",";
    strINSERT += row[2];
    Serial.println(strINSERT);
    // Create an instance of the cursor passing in the connection
    MySQL_Cursor *cur = new MySQL_Cursor(&conn);
    cur->execute(strINSERT.c_str());
    delete cur;
  }
}
```

Wow, that's a lot of code, and it's only an excerpt! I have made three sections of the code bold to bring focus to the concept I am demonstrating. Notice the get_data() method. Here we see code to read the data from the communication packet and produce a string in the form of (N,F,F,F), where N is an integer and F is a floating-point number. We use this string later in the code. The second section is the call to get_data() where we save the string created in an array. The previous section is where we detect that three data collectors have sent data and we save the information to the database with an INSERT statement.

There is one thing that using a data aggregator like this makes a bit more complicated. Recall sometimes we want to store the date and time when the data was read. If we implement a data aggregator like as shown previously where we do not capture the date and time when the values were read, setting a timestamp column in the database may result in inaccuracies in the date and time values.

For example, if it takes 10 seconds to read all three sets of data, the timestamp values will not only be about 10 seconds delayed, the three rows will have nearly the same timestamp. That may be OK for some solutions, but if the delay were more like 10 minutes or even an hour, the delay may be unacceptable.

Thus, if you want to store date and time information for data aggregators, you will have to collect that data when the sensors either are read by the data collectors or are set by the data aggregator when it receives the data.

Aggregate Calculations

The last form of aggregation you may encounter includes those situations where you need to do some calculations on a set of data. This could be as simple as calculating an average for a sum, finding minimal and maximum values, or performing any such formula or operation. You can write code to handle these operations, and that is a valid solution. However, this is another area where the database server excels.

For example, consider the code excerpt from a Python script shown in Listing 4-13. Here we see code for reading a number of rows from a file containing data with several columns. We use the power of Python to decipher the file and then perform operations on the data.

Listing 4-13. Aggregate Calculations (Python)

```
file_log = open("data_log_python.txt", 'a')
temp_tot = 0;
temp_min = 999;
temp_max = 0;
for i in range(0,20):
  # read sensors
  temp = read_sensor(1)
  baro = read_sensor(2)
  # add to total
  temp_tot = temp_tot + temp
  # find min/max
  if (temp < temp_min):
    temp_min = temp
  if (temp > temp_max):
    temp_max = temp
  print(temp, baro)
  file_log.write(strData.format(temp, baro))
# display aggregate values
print "Average Temperature:", temp_tot/20.00
print "Min Temperature:", temp_min
print "Max Temperature:", temp_max
file_log.close()
```

Notice we calculate the average of the values read (20) as well as the minimum and maximum values. There is nothing magical in the code other than that we have to count, total, and detect the min/max values. While the code is not complex, it is far more than a single line of code. An example output is shown here:

```
Average Temperature: 94.0704303696
Min Temperature: 90.2774251101
Max Temperature: 99.8600782018
```

Now let's see how to do the same operations in the database using a MySQL feature called a *function*. In this case, I used the same data for the Python code shown previously, storing it in a simple table. As you will see, doing aggregation operations on the data in a database is easy.

Listing 4-14 shows the use of special functions that you can use with the AVG, MIN, and MAX functions in the SELECT statement. These functions do exactly what you would expect. There are many more such functions. For a complete list of the functions available, see the online MySQL Reference Manual (http://dev.mysql.com/doc/refman/5.7/en/func-op-summary-ref.html).

Listing 4-14. Aggregate Calculations (SQL)

```
mysql> SELECT AVG(temperature), MIN(temperature), MAX(temperature) FROM aggregation_test;
+-------------------+-------------------+-------------------+
| AVG(temperature)  | MIN(temperature)  | MAX(temperature)  |
+-------------------+-------------------+-------------------+
| 94.0704303741455  | 90.27742767333984 | 99.86007690429688 |
+-------------------+-------------------+-------------------+
1 row in set (0.00 sec)
```

Notice we used a single command to get this data. What could be easier? Also note that the final values are slightly different than the Python example output. This is because of the imperfect rounding of floating-point values as can be seen in the greater precision decimals. For example, if we wanted to display only two decimal values, the Python and database results would be the same: 94.07, 90.28, and 99.86, respectfully.

Clearly, the statements in the database are easier to read and easier to use. All you have to do is form the query with the correct functions. You will see more about these statements in the next two chapters.

Summary

Data transformation is not just about taking one data type and changing it to another. As you saw in this chapter, there are many things you need to consider before you settle on how to do the transformation. We need to consider not only what we are observing but also what we expect to learn and how that data can be interpreted for further knowledge. We need to consider the raw data as well as any transformations needed to make the data more informative and relative.

Another consideration beyond interpreting the data is how to annotate the data. Too much annotation can obscure what we want to learn, and too little can produce false interpretations or result in missed opportunities to gain knowledge. Similarly, aggregation of data can be important if we need to combine data from multiple sensors or multiple data collectors or need to perform statistical or counting operations on the data.

In this chapter, we learned several practical questions to ask when considering the data as well as several examples of annotation and aggregation in both Arduino and Python code. This chapter also included considerations for storing IOT data in a database. In fact, we saw that many of the example annotations and aggregations may be easier to implement in the database than in code.

In the next chapter, we dive into database storage with MySQL. You will learn what MySQL is, how to install it, and how to get started with building a database for your IOT data.

■ ■ ■

MySQL Primer

So you're planning your IOT solution and have decided to build into your solution a database server. Perhaps you've never used a database system before or maybe you've used one as a user but have never had any need to set up one from scratch. Or perhaps you've decided to discover what all the fuss is about database systems in general. Whichever the case, you have the core knowledge you need to get started[1]: you know what you want to store and what the data looks like.

Recall we discussed and saw some examples of IOT data and how best to store them for an IOT solution. Recall there are trade-offs for each type of data we want to store. You also learned more about how to augment data to make it more useable in your IOT solutions. As you saw, it isn't always the case that the originator of the data (the sensor nodes or platform) that needs to do this augmentation. As you learned, data aggregators with more computing resources are better suited for such operations. However, you will also see that there is even more power in database servers (sometimes called *data nodes* since you can have more than one), which can perform data aggregation and annotation automatically.

In this chapter, you will also see how to put these techniques in practice to store data in a database. More specifically, you will learn how to use MySQL and leverage that knowledge in an IOT solution. We begin with a short discussion on how to get MySQL, install it, and make your first database. The rest of the chapter is devoted to presenting a short primer on how to use MySQL through examples.

Getting Started

MySQL is the world's most popular open source database system for many excellent reasons. First, it is open source, which means anyone can use it for a wide variety of tasks for free.[2] Best of all, MySQL is included in many platform repositories, making it easy to get and install. If your platform doesn't include MySQL in the repository (such as aptitude), you can download it from the MySQL web site (`http://dev.mysql.com`).

Oracle Corporation owns MySQL. Oracle obtained MySQL through an acquisition of Sun Microsystems, which acquired MySQL from its original owners, MySQL AB. Despite fears to the contrary, Oracle has shown excellent stewardship of MySQL by continuing to invest in the evolution and development of new features as well as faithfully maintaining its open source heritage. Although Oracle also offers commercial licenses of MySQL—just as its prior owners did in the past—MySQL is still open source and available to everyone.

---

[1]Even a rudimentary knowledge of your data and its form is crucial to a successful database configuration.
[2]According to GNU (`http://gnu.org/philosophy/free-sw.html`), "Free software is a matter of liberty, not price. To understand the concept, you should think of 'free' as in 'free speech,' not as in 'free beer.'"

WHAT IS OPEN SOURCE? IS IT REALLY FREE?

Open source software grew from a conscious resistance to the corporate-property mind-set. While working for MIT, Richard Stallman, the father of the free software movement, resisted the trend of making software private (closed) and left MIT to start the GNU (GNU Not Unix) project and the Free Software Foundation (FSF).

Stallman's goal was to reestablish a cooperating community of developers. He had the foresight, however, to realize that the system needed a copyright license that guaranteed certain freedoms. (Some have called Stallman's take on copyright "copyleft," because it guarantees freedom rather than restricts it.) To solve this, Stallman created the GNU Public License (GPL). The GPL, a clever work of legal permissions that permits the code to be copied and modified without restriction, states that derivative works (the modified copies) must be distributed under the same license as the original version without any additional restrictions.

There was one problem with the free software movement. The term *free* was intended to guarantee freedom to use, modify, and distribute; it was not intended to mean "no cost" or "free to a good home." To counter this misconception, the Open Source Initiative (OSI) formed and later adopted and promoted the phrase open source to describe the freedoms guaranteed by the GPL license. For more information about open source software, visit www.opensource.org.

How Do I Use MySQL?

MySQL runs as a background process (or as a foreground process if you launch it from the command line) on your system. Like most database systems, MySQL supports Structured Query Language (SQL). You can use SQL to create databases and objects (using data definition language [DDL]), write or change data (using data manipulation language [DML]), and execute various commands for managing the server.

To issue these commands, you must first connect to the database server. MySQL provides a client application named mysql[3] that enables you to connect to and run commands on the server. The client accepts SQL commands as well as a few commands specific to the client itself. A semicolon must terminate all commands.

■ **Tip** To see a list of the commands available in the client, type **help** and press Enter at the prompt.

To connect to the server, you must specify a user account and the server to which you want to connect. If you are connecting to a server on the same machine, you can omit the server information (host and port) as these default to localhost on port 3306. The user is specified using the --user (or -u) option. You can specify the password for the user on the command, but the more secure practice is to specify --password (or -p), and the client with prompt you for the password. If you do specify the password on the command line, you will be prompted with a warning encouraging you to not use that practice.

Using the mysql client on the same machine without the --host (or -h) and --port option does not use a network connection. If you want to connect using a network connection or want to connect using a different port, you must use the loopback address. For example, to connect to a server running on port 13001 on the same machine, use the command mysql -uroot -p –h127.0.0.1 --port=13001.

---

[3]Sometimes called the MySQL monitor, terminal monitor, or even the MySQL command window.

Listing 5-1 shows examples of several SQL commands in action using the mysql client.

Listing 5-1. Commands Using the mysql Client

```
$ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.8-rc-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE plant_monitoring;
Query OK, 1 row affected (0.01 sec)

mysql> CREATE TABLE plant_monitoring.plants (plant_name char(50), sensor_value int,
sensor_event timestamp);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO plant_monitoring.plants VALUES ('living room', 23, NULL);
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM plant_monitor.plants;
+-------------+--------------+---------------------+
| plant_name  | sensor_value | sensor_event        |
+-------------+--------------+---------------------+
| living room |           23 | 2015-09-22 19:54:01 |
+-------------+--------------+---------------------+
1 row in set (0.01 sec)

mysql> SET @@global.server_id = 111;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

In this example, you see DML in the form of the CREATE DATABASE and CREATE TABLE statements, DDL in the form of the INSERT and SELECT statements, and a simple administrative command to set a global server variable. Next you see the creation of a database and a table to store the data, the addition of a row in the table, and finally the retrieval of the data in the table. Notice how I used capital letters for SQL command keywords. This is a common practice and helps make the SQL commands easier to read and easier for find user-supplied options or data.

You can exit the MySQL client by typing the command quit. On Linux and Unix systems, you can press Ctrl+D to exit the client.

A great many commands are available in MySQL. Fortunately, you need master only a few of the more common ones. The following are the commands you will use most often. The portions enclosed in <> indicate user-supplied components of the command, and [...] indicates that additional options are needed.

- `CREATE DATABASE <database_name>`: Creates a database

- `USE <database>`: Sets the default database (not an SQL command)

- `CREATE TABLE <table_name> [...]`: Creates a table or structure to store data

- `INSERT INTO <table_name> [...]`: Adds data to a table

- `UPDATE [...]`: Changes one or more values for a specific row

- `DELETE FROM <table_name> [...]`: Removes data from a table

- `SELECT [...]`: Retrieves data (rows) from the table

- `SHOW [...]`: Shows a list of the objects

Although this list is only a short introduction and nothing like a complete syntax guide, there is an excellent online reference manual that explains every command (and much more) in great detail. You should refer to the online reference manual whenever you have a question about anything in MySQL. You can find it at http://dev.mysql.com/doc/.

One of the more interesting commands shown allows you to see a list of objects. For example, you can see the databases with `SHOW DATABASES`, a list of tables (once you change to a database) with `SHOW TABLES`, and even the permissions for users with `SHOW GRANTS`. I find myself using these commands quite frequently.

■ **Tip**   If you use the mysql client, you must terminate each command with a semicolon (`;`) or `\G`.

If you are thinking that there is a lot more to MySQL than a few simple commands, you are absolutely correct. Despite its ease of use and fast startup time, MySQL is a full-fledged relational database management system (RDBMS). There is much more to it than you've seen here. For more information about MySQL, including all the advanced features, see the reference manual.

MYSQL—WHAT DOES IT MEAN?

The name MySQL is a combination of a proper name and an acronym. SQL is Structured Query Language. The *My* part isn't the possessive form—it is a name. In this case, *My* is the name of the founder's daughter. As for pronunciation, MySQL experts pronounce it "My-S-Q-L" and not "my sequel."

How to Get and Install MySQL

The MySQL server is available for a variety of platforms including most Linux and Unix platforms, Mac OS X, and Windows. To download MySQL server, visit http://dev.mysql.com/downloads/ and click Community and then MySQL Community Server. This is the GPLv2 license of MySQL.[4] The page will automatically detect your operating system. If you want to download for another platform, you can select it from the drop-down list.

---

[4]If you are a paid customer of Oracle and have a subscription or support agreement for MySQL, contact your sales representative for details.

The download page will list several files for download. Depending on your platform, you may see several options including compressed files, source code, and installation packages. Most will choose the installation package for installation on a laptop or desktop computer. Figure 5-1 shows an example for the APT repository for the Debian and Ubuntu platforms.


Select Platform:
Ubuntu Linux

Install Using APT:

MySQL APT Repository

Supported Platforms:
▶ Debian
▶ Ubuntu

Download Now »

Download Packages:

Ubuntu Linux 15.04 (x86, 32-bit), DEB Bundle MySQL Server	5.7.8	175.6M	**Download**
(mysql-server_5.7.8-rc-1ubuntu15.04_i386.deb-bundle.tar)	MD5: 1ad157fe528d5d236cbf21f7f85f8e04	Signature	
Ubuntu Linux 15.04 (x86, 64-bit), DEB Bundle MySQL Server	5.7.8	178.4M	**Download**
(mysql-server_5.7.8-rc-1ubuntu15.04_amd64.deb-bundle.tar)	MD5: 75fb3a11ecf4a130bbf60a23554d85b3	Signature	

Figure 5-1. Download page for Ubuntu Linux
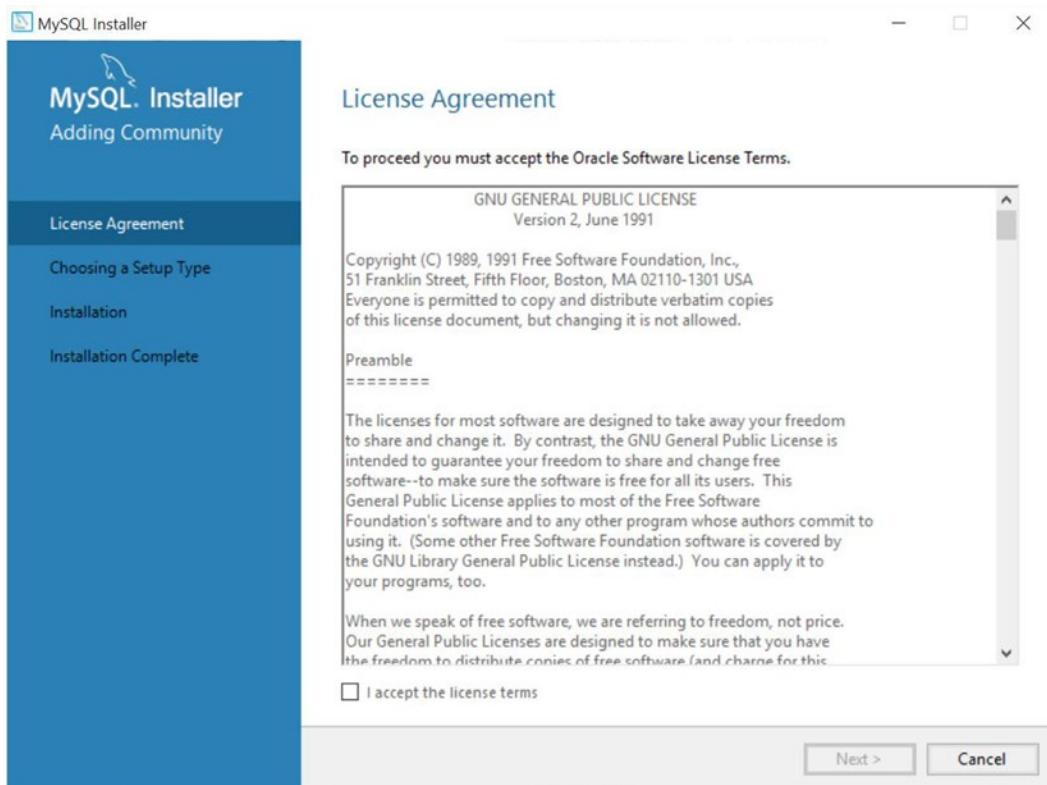
One of the most popular platforms is Microsoft Windows. Oracle has provided a special installation packaging for Windows named the Windows Installer. This package includes all the MySQL products available under the community license including MySQL Server, Workbench, Utilities, Fabric, and all of the available connectors (program libraries for connecting to MySQL). This makes installing on Windows a one-stop, one-installation affair. Figure 5-2 shows the download page for the Windows installer. The following paragraphs demonstrate how to install MySQL on Windows 10. You will see how to install MySQL on single-board computers like the Raspberry Pi in the next chapter.

Figure 5-2. *Download page for Windows Installer*

Begin by choosing either the Windows Installer 32- or 64-bit installation package that matches your Windows version. Once the file is downloaded, click the file to begin installation. Note that some browsers such as the new Edge browser may ask you if you want to launch the installation. You may need to reply to a dialog permitting the installation.
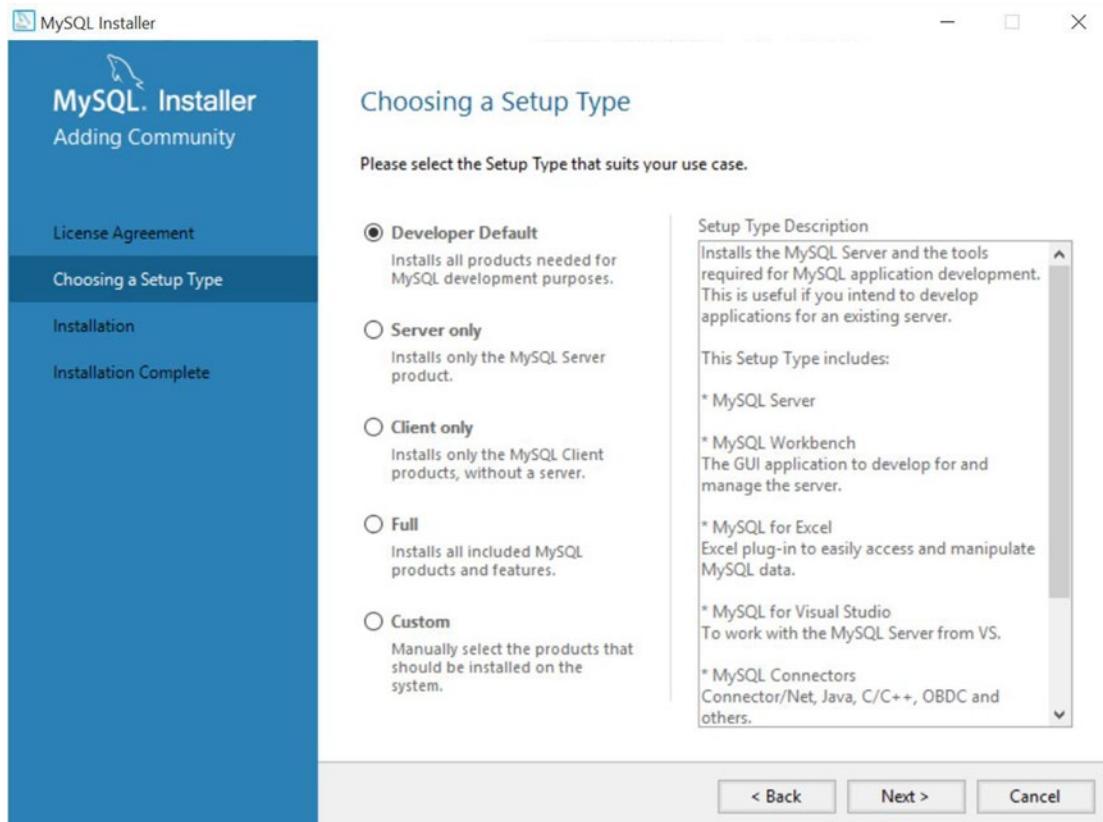
The first step is agreeing to the license. Figure 5-3 shows the license agreement panel of the installation dialog.

Figure 5-3. *License agreement*

The license shown is the GPLv2 license for the community edition. Once you have read the license[5] and agree, select the "I accept the license terms" checkbox and click Next.
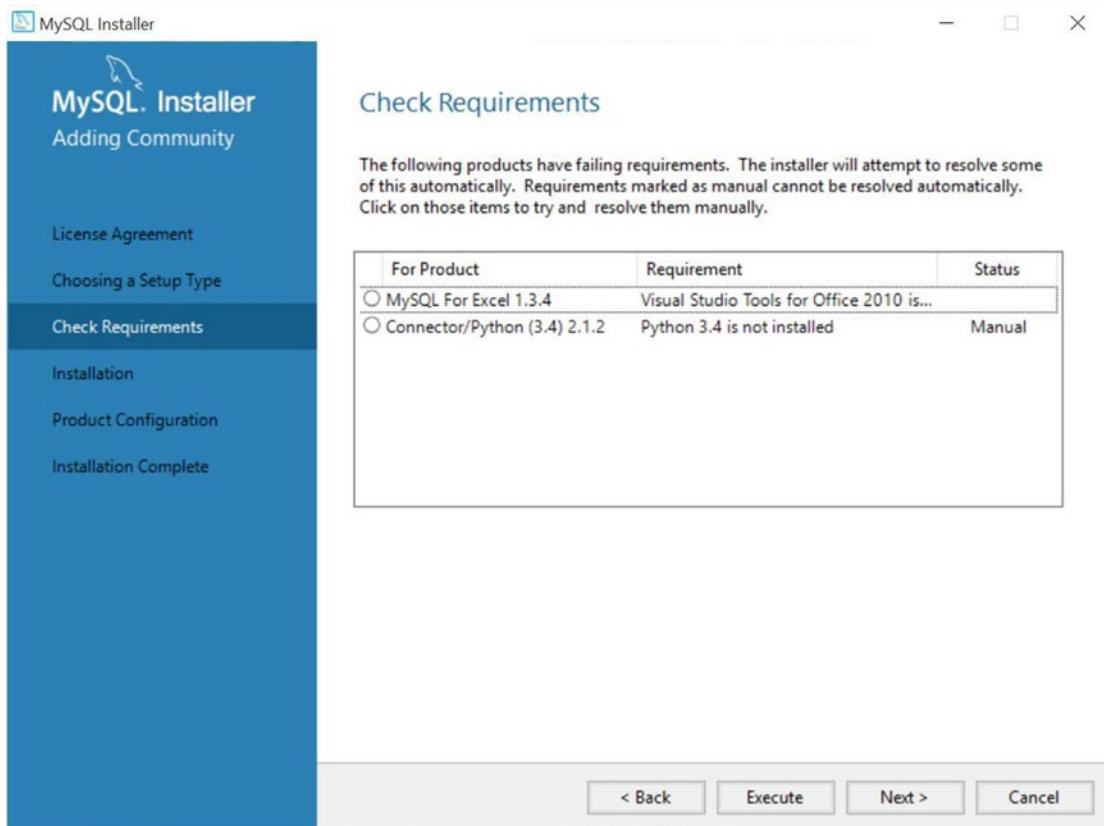
The next panel displays the setup or installation type. Most will choose the developer option because it installs all the MySQL components and applications and sets the defaults for running MySQL on the local machine. You can choose a different option and read more about each in the text to the right. Figure 5-4 shows the setup type panel. Once you make a selection, click Next.

Figure 5-4. *Setup type*
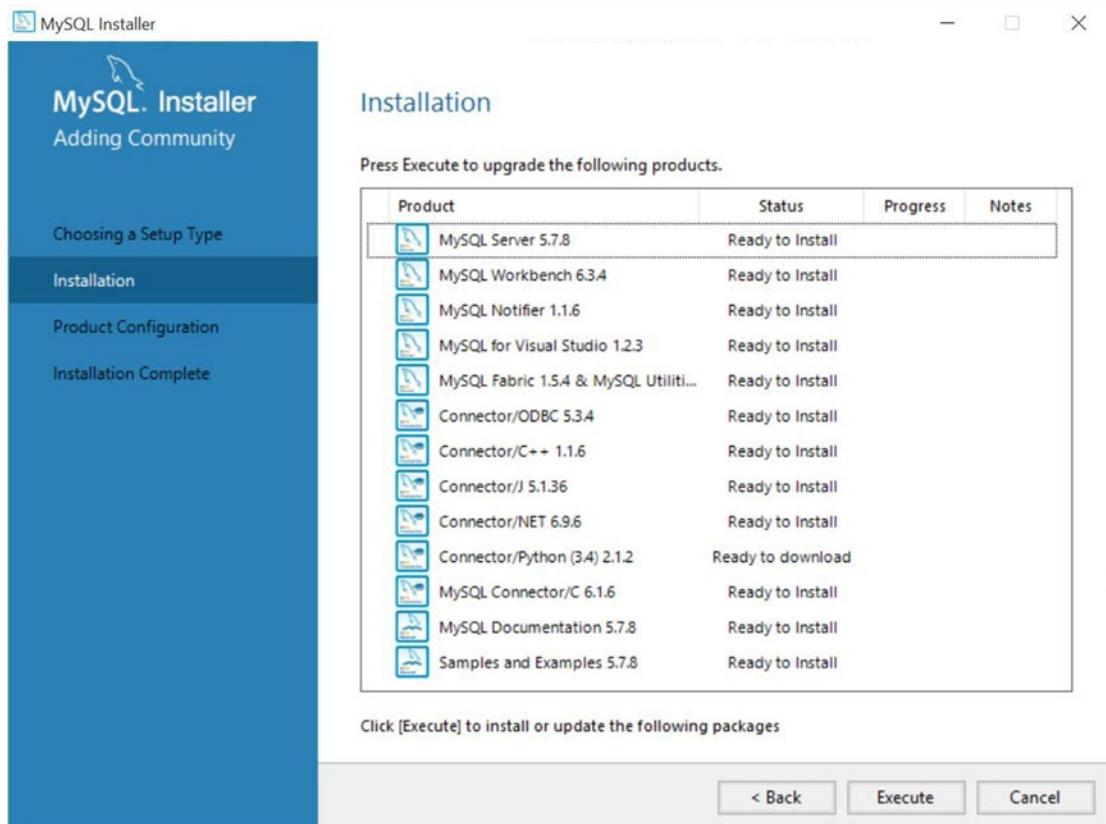
The next panel will check for any required components. For example, if you are installing on a machine that does not include Python or Visual Studio, you will get a warning, as shown in Figure 5-5. To proceed, you must resolve each of the issues. That is, the buttons will not be available until the requirement is resolved. Once you have them resolved, click Next.

---

[5]You do read these, don't you?
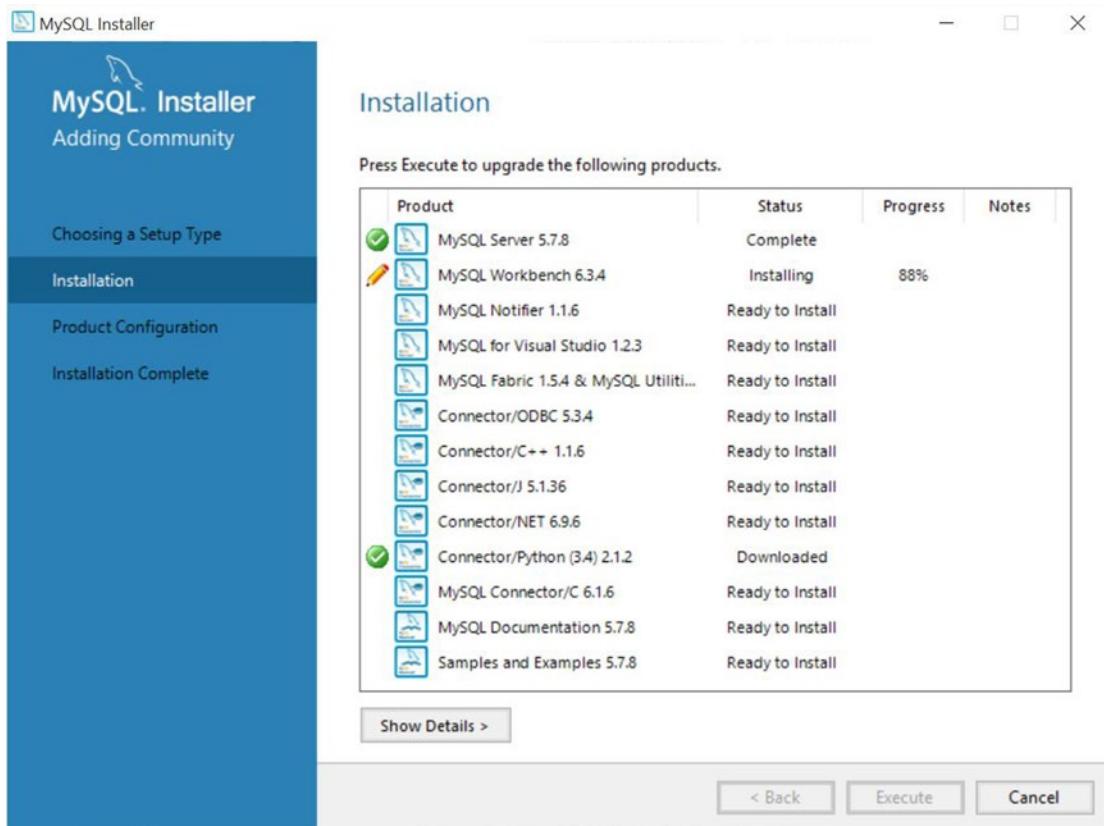
Figure 5-5. *Check Requirements page*

The next panel will show all the packages available for installation. If you do not want to install one or more of them, you can click each and choose not to install them. Figure 5-6 shows an example with all packages marked for installation. Once you are satisfied with the options for installation, click Execute.

Figure 5-6. Preparing for installation
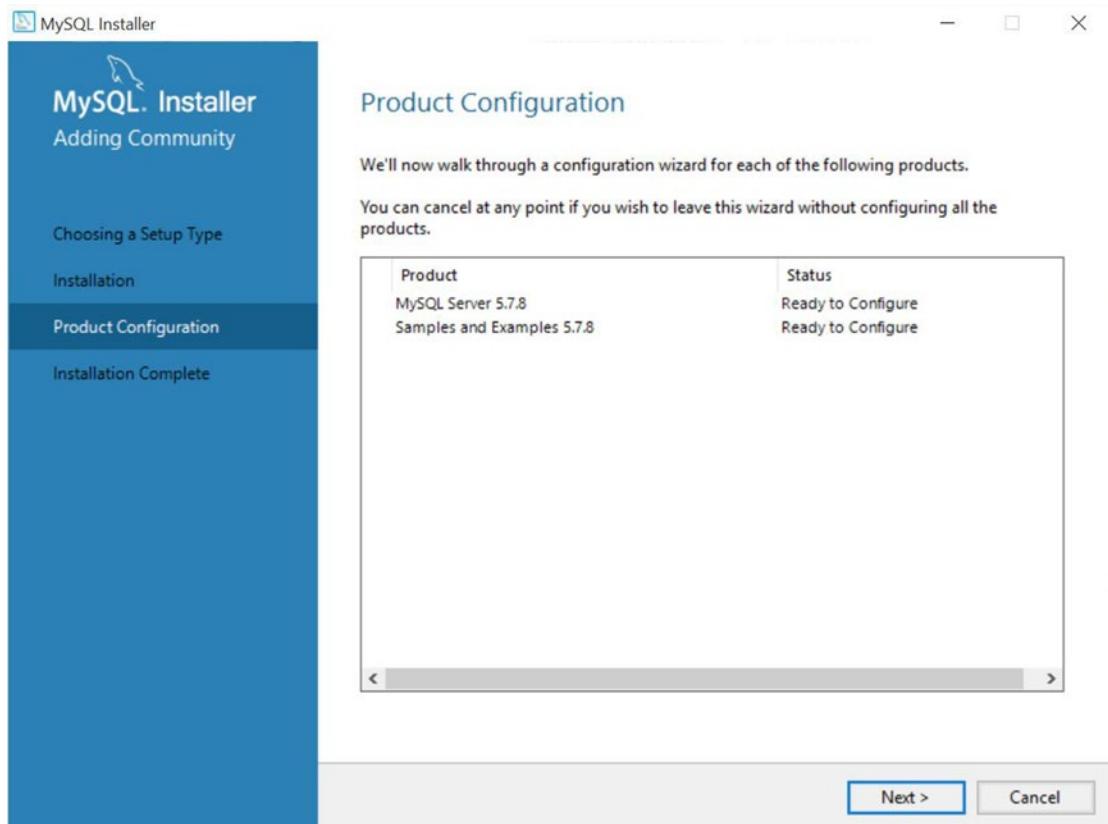
The panel will update with the progress of each installation, as shown in Figure 5-7.

Figure 5-7. *Installing packages*
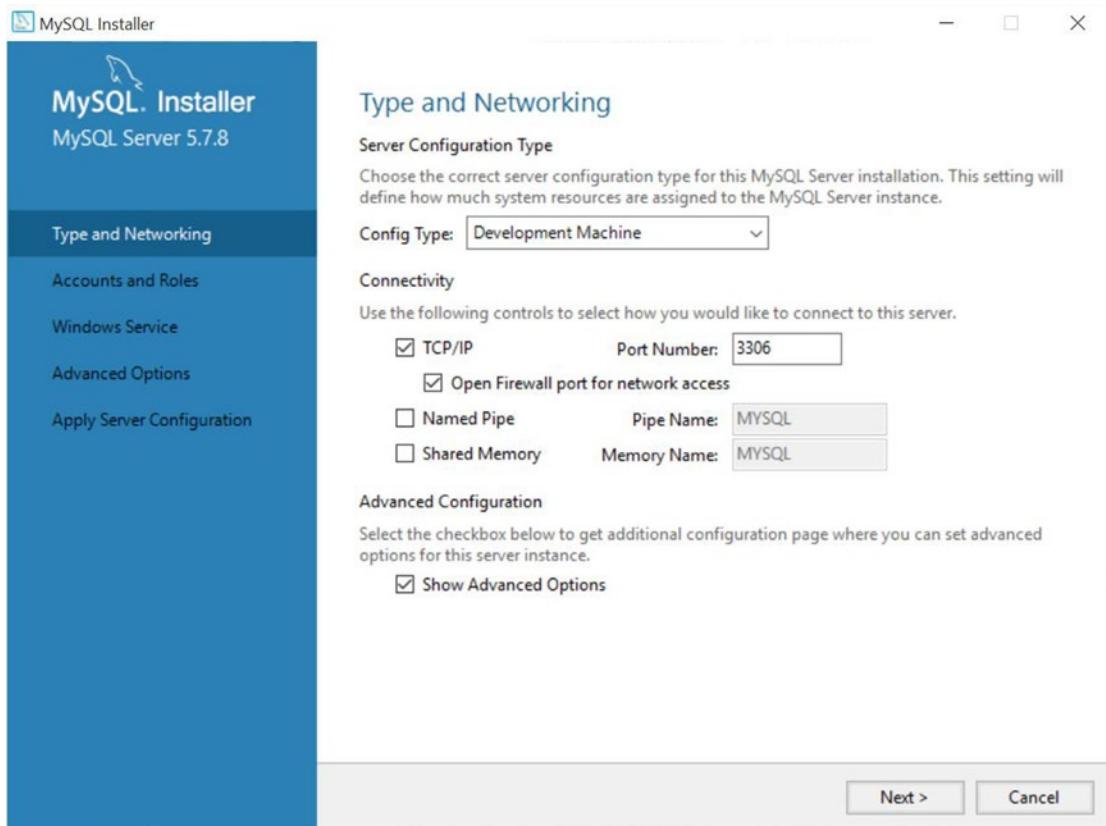
Different symbols will appear next to each package as the installation proceeds. In some cases, the installer may download additional packages. You can see this in Figure 5-7. Note that Connector/Python 2.1.2 was downloaded. You can click the Show Details button to see more details of the installation.

Once all the packages selected have been installed, you will see the configuration panel, as shown in Figure 5-8. This panel shows you a list of the different configuration options based on the packages you chose. If you chose to install everything, you will see a panel similar to the figure. Click Next to proceed.
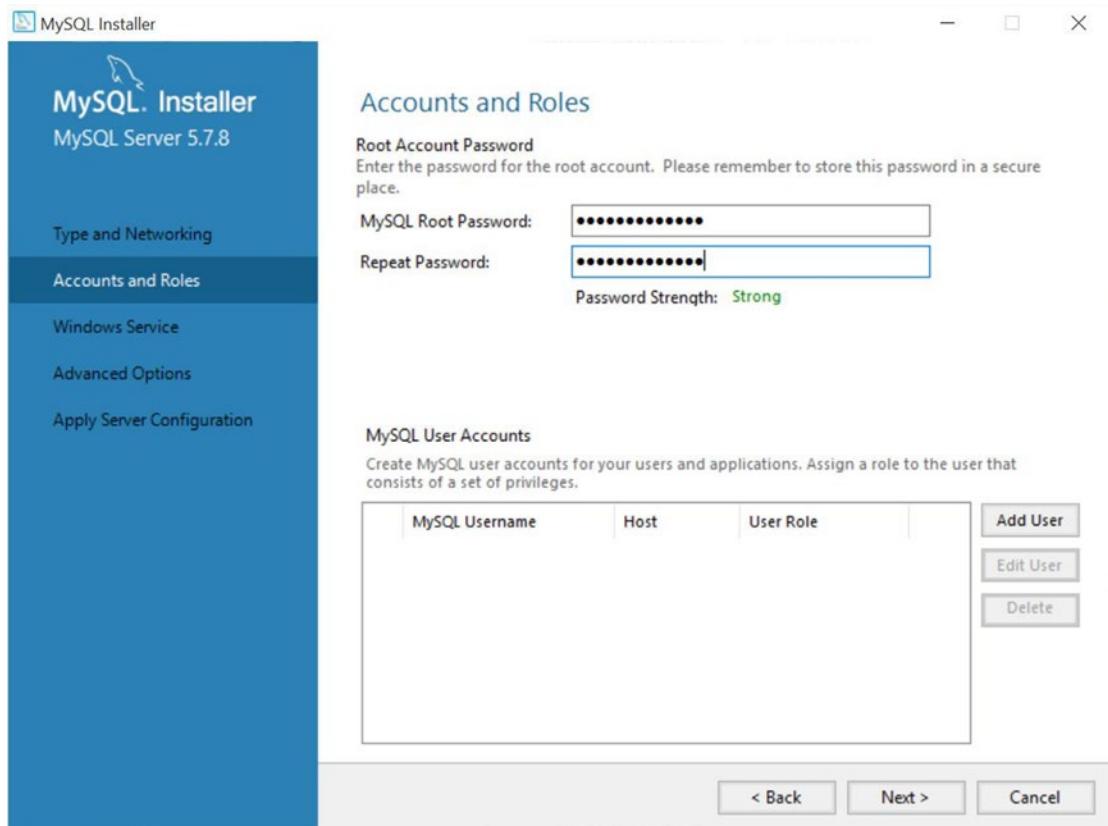
Figure 5-8. *Product configuration*

The configuration process will be different for each product. For example, to configure MySQL server, there are several steps beginning with networking, as shown in Figure 5-9. This panel allows you to choose a configuration type (how the server launches and runs) as well as the networking specifics. It is on this page that you can choose the TCP/IP port that the server will listen for connections. If you want to configure additional parameters, select the Show Advanced Options checkbox. Once you have made you selections, click Next to move to the next step.
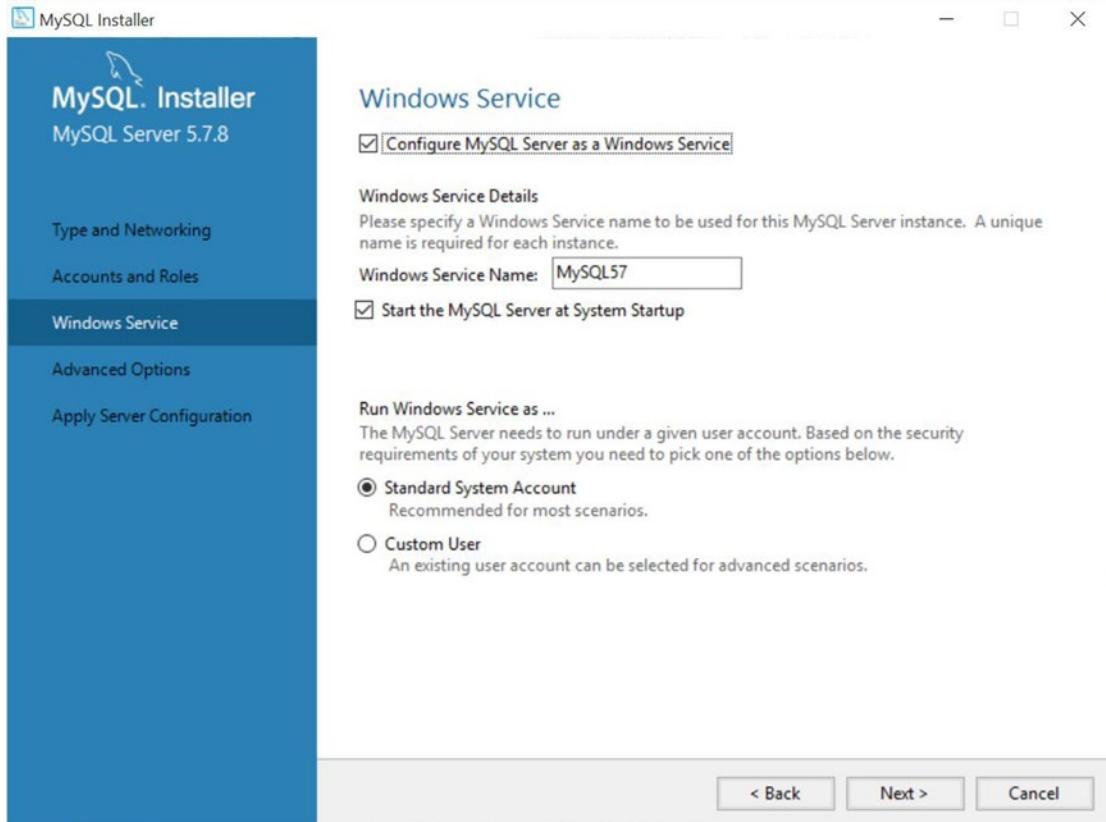
Figure 5-9. *Type and Networking page*

The next panel is the accounts and roles panel, as shown in Figure 5-10. This panel allows you to set up initial user accounts as well as the root password. It is strongly recommended you select a strong password for the root account. You can also set up additional user accounts with different roles by clicking the Add User button. Once you have your settings chosen, click Next to move to the next step.
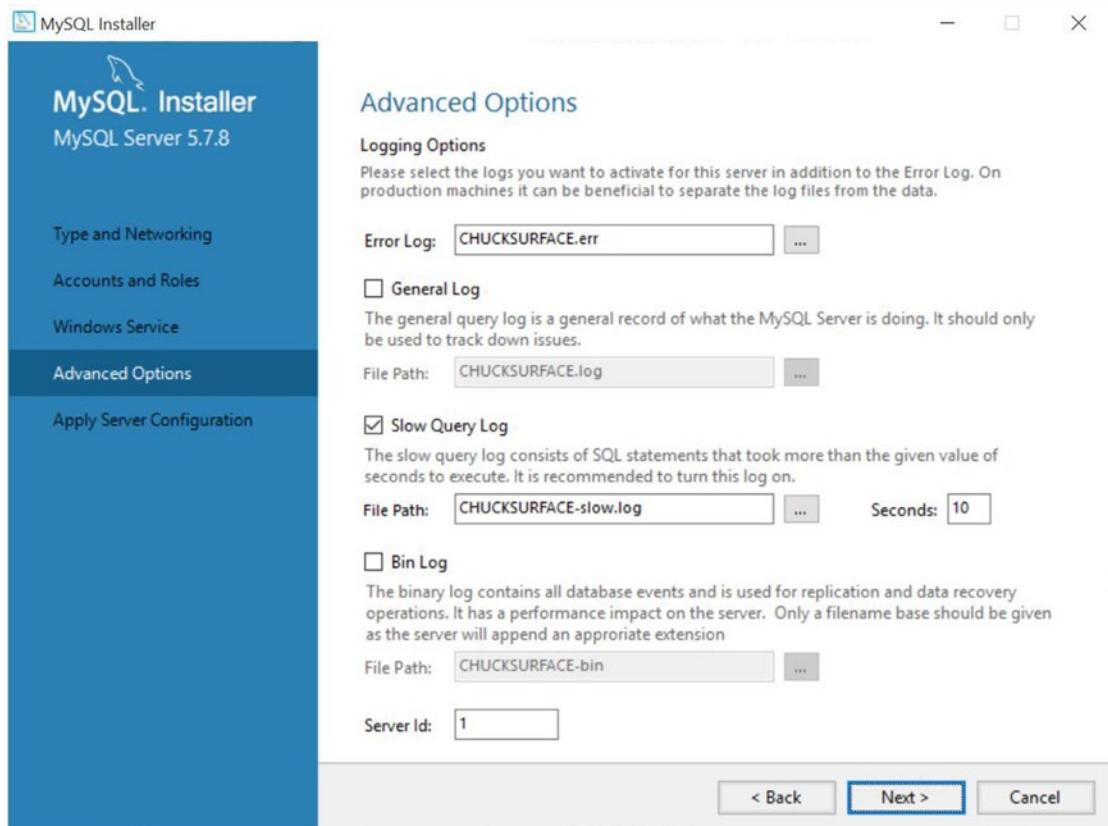
Figure 5-10. *Accounts and Roles page*

The next panel allows you to control how MySQL is started on the Windows machine. Figure 5-11 shows the details. Notice you can configure the server to start as a Windows service, start MySQL automatically at startup (or not), and what type of account the server will use. I strongly recommend leaving the default for that setting unless you know how to setup an account for running a service. Click Next to move to the next step.
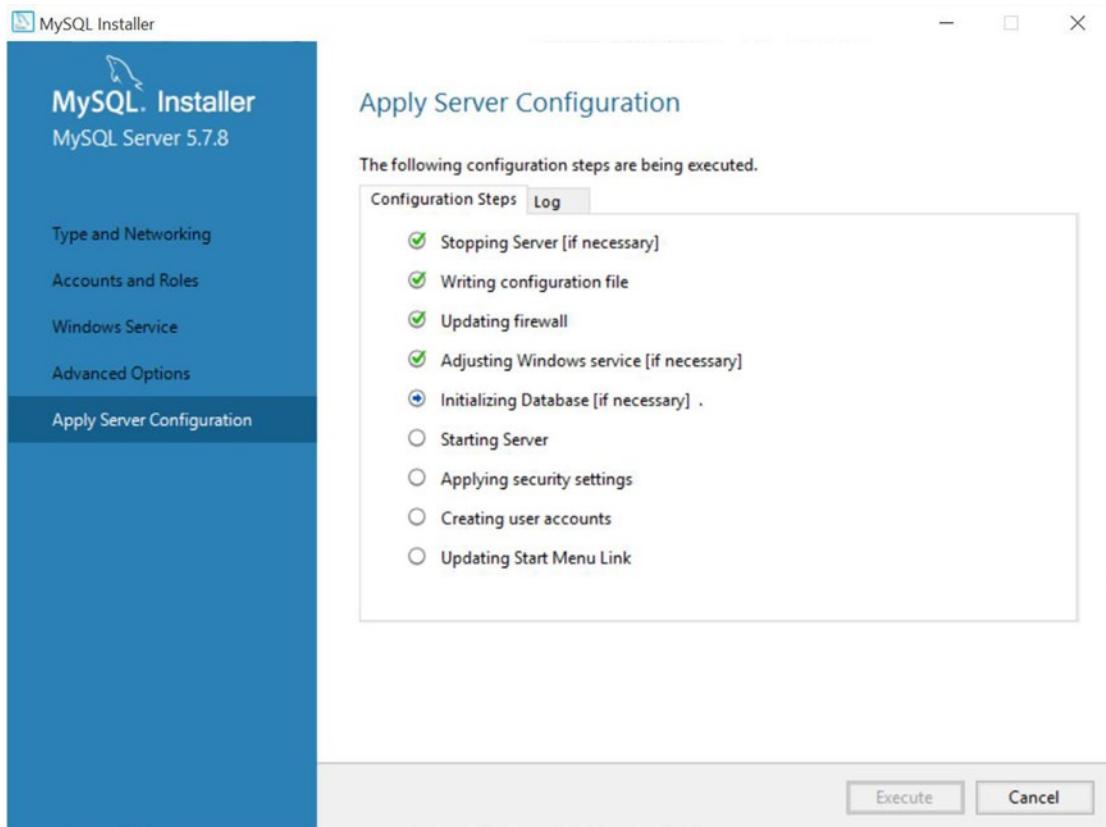
Figure 5-11. *Windows Service page*

If you checked the advanced settings in the first configuration panel, you will see the advanced options panel, as shown in Figure 5-12. This panel allows you to turn on the general log (for recording server feedback statements), query log (for recording all queries), and the binary log (for use in replication and backup). If you plan to use the server in a replication setup (I will discuss this in Chapter 7), you should turn on the binary log. The server ID must be unique among all servers in a replication setup, and you can set that on this panel. Once you have chosen your settings, click Next.
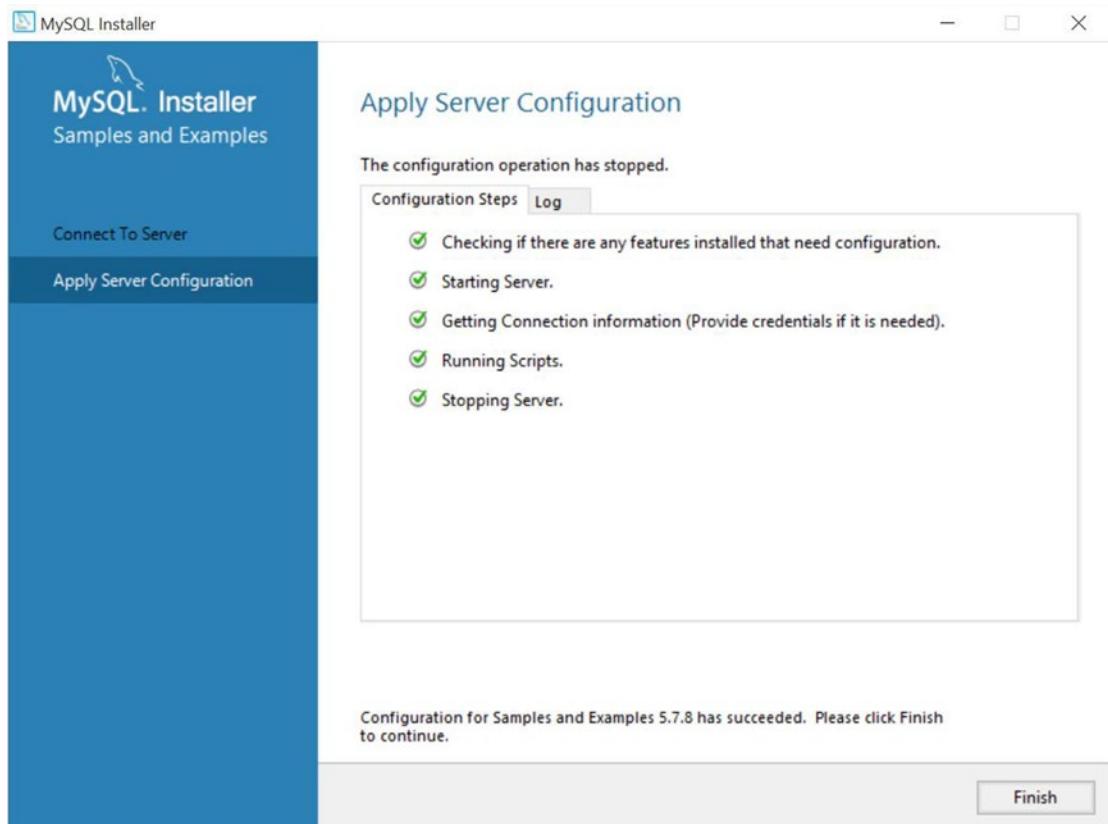
Figure 5-12. *Advanced Options panel*

The next panel shown in Figure 5-13 displays the progress of the configuration.

Figure 5-13. *Server configuration execution*

If you chose to start MySQL as a Windows service, you will see a second set of statements listed, as shown in Figure 5-14. Finally, if you chose to install the samples and example databases, you will see another dialog panel showing the progress of installing the sample databases. Once all steps are complete, click Finish.

Figure 5-14. *Configuring examples*

One of the things I like most about the Windows Installer besides being a one-stop installation mechanism is the ability to use the installer again to make changes. That is, you can run the installer another time to install a different package or even remove packages you no longer need. It is a handy way to install and configure MySQL on Windows.

■ **Note** Installers for other platforms are adopting similar mechanisms as the Windows Installer. For example, most have the configuration steps in the installation package.

Now that you know how to set up MySQL, let's discuss how you use MySQL to store and retrieve data.

How Data Is Stored and Retrieved

Now that you know what MySQL is and how it is used, you need to know a bit more about RDBMSs and MySQL in particular before you start building your first database server. This section discusses how MySQL stores data (and where it is stored), how it communicates with other systems, and some basic administration tasks required to manage your new MySQL server.

■ **Note**   I will show you how to install MySQL on the Raspberry Pi and similar boards in Chapter 6.

WHAT IS A RELATIONAL DATABASE MANAGEMENT SYSTEM?

An RDBMS is a data storage and retrieval service based on the Relational Model of Data as proposed by E. F. Codd in 1970. These systems are the standard storage mechanism for structured data. A great deal of research is devoted to refining the essential model proposed by Codd, as discussed by C. J. Date in *The Database Relational Model: A Retrospective Review and Analysis.*[6] This evolution of theory and practice is best documented in *The Third Manifesto.*[7]

The relational model is an intuitive concept of a storage repository (database) that can be easily queried by using a mechanism called a *query language* to retrieve, update, and insert data. Many vendors have implemented the relational model because it has a sound systematic theory, a firm mathematical foundation, and a simple structure. The most commonly used query mechanism is SQL, which resembles natural language. Although SQL is not included in the relational model, it provides an integral part of the practical application of the relational model in RDBMSs.

The data are represented as related pieces of information (attributes or columns sometimes called *fields*) about a certain event or entity. The set of values for the attributes is formed as a *tuple* (sometimes called a *record* or *row*). Tuples are stored in tables that have the same set of attributes. Tables can then be related to other tables through constraints on keys, attributes, and tuples.

Tables can have special mappings of columns called *indexes* that permit you to read the data in a specific order. Indexes are also useful for fast retrieval of rows that match the value(s) of the indexed columns.

How and Where MySQL Stores Data

The MySQL database system stores data via an interesting mechanism of programmatic isolation called a *storage engine* that is governed by the handler interface. The handler interface permits the use of interchangeable storage components in the MySQL server so that the parser, the optimizer, and all manner of components can interact in storing data on disk using a common mechanism. This is also referred to as a *pluggable* storage engine.

■ **Note**   MySQL supports several storage engines. Most are designed to write data to disk by default. However, the MEMORY storage engine stores data in memory but is not persistent. That is, when the computer is rebooted, the data is lost. You can use the MEMORY storage engine for fast lookup tables. Indeed, one optimization technique is to create copies of lookup tables at startup using the MEMORY storage engine.

---

[6]C. J. Date, *The Database Relational Model: A Retrospective Review and Analysis* (Reading, MA: Addison-Wesley, 2001).
[7]C. J. Date and H. Darwen, *Foundation for Future Database Systems: The Third Manifesto* (Reading, MA: Addison-Wesley, 2000).

What does this mean to you? It means you have the choice of different mechanisms for storing data. You can specify the storage engine in the table CREATE statement shown in the following code sample. Notice the last line in the command: this is how a storage engine is specified. Leaving off this clause results in MySQL using the default storage engine. For the examples in this book, MySQL 5.5 uses the MyISAM storage engine by default.

■ **Tip**   The default storage engine was changed from MyISAM to InnoDB in MySQL version 5.6.

```
CREATE DATABASE `bvm`;

CREATE TABLE `bvm`.`books` (
  `ISBN` varchar(15) DEFAULT NULL,
  `Title` varchar(125) DEFAULT NULL,
  `Authors` varchar(100) DEFAULT NULL,
  `Quantity` int(11) DEFAULT NULL,
  `Slot` int(11) DEFAULT NULL,
  `Thumbnail` varchar(100) DEFAULT NULL,
  `Description` text
) ENGINE=MyISAM;
```

Great! Now, what storage engines exist on MySQL? You can discover which storage engines are supported by issuing the SHOW STORAGE ENGINES command, as shown in Listing 5-2. As you see, there are a lot to choose from. I cover a few that may be pertinent to planning IOT solutions.

■ **Note**   The following sections show how to work with MySQL on a typical Linux-like (actually Unix-like) platform. I've found most IOT solutions will use forms of these platforms rather than Windows 10, but that may change in the future. For now, I focus on exploring MySQL on these platforms rather than Windows. However, many of the examples shown can be executed on Windows albeit with a different set of commands.

Listing 5-2. Available Storage Engines

```
mysql> SHOW STORAGE ENGINES \G
*************************** 1. row ***************************
      Engine: FEDERATED
     Support: NO
     Comment: Federated MySQL storage engine
Transactions: NULL
          XA: NULL
  Savepoints: NULL
*************************** 2. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
```

```
*************************** 3. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 7. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 9. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
9 rows in set (0.00 sec)

mysql>
```

As of version 5.6, MySQL uses the InnoDB storage engine by default. Previous versions used MyISAM as the default. InnoDB is a fully transactional, ACID[8] storage engine. A transaction is a batch of statements that must all succeed before any changes are written to disk. The classic example is a bank transfer. If you consider a system that requires deducting an amount from one account and then crediting that amount to another account to complete the act of moving funds, you would not want the first to succeed and the second to fail, or vice versa!

Wrapping the statements in a transaction ensures that no data is written to disk until and unless all statements are completed without errors. Transactions in this case are designated with a BEGIN statement and concluded with either a COMMIT to save the changes or a ROLLBACK to undo the changes. InnoDB stores its data in a single file (with some additional files for managing indexes and transactions).

The MyISAM storage engine is optimized for reads. MyISAM has been the default for some time and was one of the first storage engines available. In fact, a large portion of the server is dedicated to supporting MyISAM. It differs from InnoDB in that it does not support transactions and stores its data in an indexed sequential access method format. This means it supports fast indexing. You would choose MyISAM over InnoDB if you did not need transactions and you wanted to be able to move or back up individual tables.

Another storage engine that you may want to consider, especially for sensor networks, is Archive. This engine does not support deletes (but you can drop entire tables) and is optimized for minimal storage on disk. Clearly, if you are running MySQL on a small system like a Raspberry Pi, small is almost always better! The inability to delete data may limit more advanced applications, but most sensor networks merely store data and rarely delete it. In this case, you can consider using the Archive storage engine.

There is also the CSV storage engine (where CSV stands for comma-separated values). This storage engine creates text files to store the data in plain text that can be read by other applications such as a spreadsheet application. If you use your sensor data for statistical analysis, the CSV storage engine may make the process of ingesting the data easier.

So, where is all this data? If you query the MySQL server and issue the command SHOW VARIABLES LIKE "datadir";, you see the path to the location on disk that all storage engines use to store data. In the case of InnoDB, this is a single file on disk located in the data directory. InnoDB also creates a few administrative files, but the data is stored in the single file. For most other storage engines except NDB and MEMORY, the data for the tables is stored in a folder with the name of the database under the data directory. Listing 5-3 shows an example from a Mac OS X machine. You may need to use different paths on your own machine.

Listing 5-3. Finding Where Your Data Is Located

```
mysql> SHOW VARIABLES LIKE 'datadir';
+---------------+-----------------------+
| Variable_name | Value                 |
+---------------+-----------------------+
| datadir       | /usr/local/mysql/data/ |
+---------------+-----------------------+
1 row in set (0.00 sec)

mysql> quit;
bye

$ sudo ls -lsa /usr/local/mysql/data
rwxr-x---   58 _mysql  wheel        1972 Feb  6 15:05 .
drwxr-xr-x  17 root    wheel         578 Jan 20 16:38 ..
-rw-rw----   1 _mysql  wheel           0 Feb  6 15:04 Chucks-iMac.local.err
-rw-rw----   1 _mysql  wheel           5 Feb  6 15:00 Chucks-iMac.local.pid
```

---

[8] http://en.wikipedia.org/wiki/ACID

```
drwx------     6 _mysql  wheel        204 Oct 17 15:16 bvm
-rw-rw----     1 _mysql  wheel    5242880 Feb  6 15:00 ib_logfile0
-rw-rw----     1 _mysql  wheel    5242880 Feb  6 15:00 ib_logfile1
-rw-rw----     1 _mysql  wheel  815792128 Feb  1 17:16 ibdata1
-rw-rw----     1 _mysql  wheel   52428800 Feb  1 17:16 ibdata2
drwxr-x---    77 _mysql  wheel       2618 Jan  8 15:24 mysql
drwx------    38 _mysql  wheel       1292 Nov 27 08:46 sakila
drwx------   192 _mysql  wheel       6528 Oct 22 12:17 test
drwx------     6 _mysql  wheel        204 Dec 18 17:05 world_innodb

$ sudo ls -lsa /usr/local/mysql/data/bvm
drwx------     6 _mysql  wheel   204 Oct 17 15:16 .
drwxr-x---    58 _mysql  wheel  1972 Feb  6 15:05 ..
-rw-rw----     1 _mysql  wheel  5056 Oct 17 15:24 books.MYD
-rw-rw----     1 _mysql  wheel  1024 Oct 17 15:25 books.MYI
-rw-rw----     1 _mysql  wheel  8780 Oct 17 15:16 books.frm
-rw-rw----     1 _mysql  wheel    65 Oct 17 15:15 db.opt
```

This example first queries the database server for the location of the data directory (it is in a protected folder on this machine). If you issue a listing command, you can see the InnoDB files identified by the ib and ibd prefixes. You also see a number of directories, all of which are the databases on this server. After that is a listing of one of the database folders. Notice the files with the extension .MY?: these are MyISAM files (data and index). The .frm files are the configuration files created and maintained by the server.

■ **Tip** If you want to copy data from one server to another by copying files, be sure to copy the .frm files as well! This is easy for MyISAM and Archive but much harder with InnoDB. In the case of InnoDB, you have to copy all the database folders and the InnoDB files to make sure you get everything.

Although it is unlikely that you would require a transactional storage engine for a database node in your IOT solution, such as a Raspberry Pi running MySQL Server, MySQL 5.6 has one, and it's turned on by default. A more likely scenario is that you would use the MyISAM or Archive engine for your tables.

For more information about storage engines and the choices and features of each, please see the online MySQL Reference Manual section "Storage Engines" (http://dev.mysql.com/doc/).

The MySQL Configuration File

The MySQL server can be configured using a configuration file similar to the way you configure the Raspberry Pi. On Windows, the MySQL configuration file is located in the installation folder and is named my.ini. On other systems, it is located in the /etc/mysql folder and is named my.cnf. This file contains several sections, one of which is labeled [mysqld]. The items in this list are key-value pairs; the name on the left of the equal sign is the option, and its value on the right. The following is a typical configuration file (with many lines suppressed for brevity):

```
[mysqld]
port = 3306
basedir = /usr/local/mysql
datadir = /usr/local/mysql/data
server_id = 5
general_log
```

As you can see, this is a simple way to configure a system. This example sets the TCP port, base directory (the root of the MySQL installation including the data as well as binary and auxiliary files), data directory, and server ID (used for replication, as discussed shortly) and turns on the general log (when the Boolean switch is included, it turns on the log). There are many such variables you can set for MySQL. See the online MySQL reference manual for details concerning using the configuration file. You will change this file when you set up MySQL on the Raspberry Pi.

How to Start, Stop, and Restart MySQL on Windows

While working with your databases and configuring MySQL on your computer, you may need to control the startup and shutdown of the MySQL server. The default mode for installing MySQL is to automatically start on boot and stop on shutdown, but you may want to change that, or you may need to stop and start the server after changing a parameter. In addition, when you change the configuration file, you need to restart the server to see the effect of your changes.

You can start, stop, and restart the MySQL server with the notifier tray application or via the Windows services control panel. Simply select the MySQL service and right-click to stop or start the service. This will execute a controlled shutdown and startup should you need to do so.

Creating Users and Granting Access

You need to know about two additional administrative operations before working with MySQL: creating user accounts and granting access to databases. MySQL can perform both of these with the GRANT statement, which automatically creates a user if one does not exist. But the more pedantic method is first to issue a CREATE USER command followed by one or more GRANT commands. For example, the following shows the creation of a user named sensor1 and grants the user access to the database room_temp:

```
CREATE USER 'sensor1'@'%' IDENTIFIED BY 'secret';
GRANT SELECT, INSERT, UPDATE ON room_temp.* TO 'sensor1'@'%';
```

The first command creates the user named sensor1, but the name also has an @ followed by another string. This second string is the host name of the machine with which the user is associated. That is, each user in MySQL has both a user name and a host name, in the form user@host, to uniquely identify them. That means the user and host sensor1@10.0.1.16 and the user and host sensor1@10.0.1.17 are not the same. However, the % symbol can be used as a wildcard to associate the user with any host. The IDENTIFIED BY clause sets the password for the user.

A NOTE ABOUT SECURITY

It is always a good idea to create a user for your application that does not have full access to the MySQL system. This is so you can minimize any accidental changes and also to prevent exploitation. For sensor networks, it is recommended that you create a user with access only to those databases where you store (or retrieve) data. You can change MySQL user passwords with the following command:

```
SET PASSWORD FOR sensor1@"%" = PASSWORD("secret");
```

Also be careful about using the wildcard % for the host. Although it makes it easier to create a single user and let the user access the database server from any host, it also makes it much easier for someone bent on malice to access your server (once they discover the password).

> Another consideration is connectivity. As with the Raspberry Pi, if you connect a database to your network and the network is in turn connected to the Internet, it may be possible for other users on your network or the Internet to gain access to the database. Don't make it easy for them—change your root user password, and create users for your applications.

The second command allows access to databases. There are many privileges that you can give a user. The example shows the most likely set that you would want to give a user of a sensor network database: read (SELECT), add data (INSERT), and change data (UPDATE). See the online reference manual for more about security and account access privileges.

The command also specifies a database and objects to which to grant the privilege. Thus, it is possible to give a user read (SELECT) privileges to some tables and write (INSERT, UPDATE) privileges to other tables. This example gives the user access to all objects (tables, views, and so on) in the room_temp database.

As mentioned, you can combine these two commands into a single command. You are likely to see this form more often in the literature. The following shows the combined syntax. In this case, all you need to do is add the IDENTIFIED BY clause to the GRANT statement. Cool!

```
GRANT SELECT, INSERT, UPDATE ON room_temp.* TO  'sensor1'@'%' IDENTIFIED BY 'secret';
```

Common MySQL Commands and Concepts

Learning and mastering a database system requires training, experience, and a good deal of perseverance. Chief among the knowledge needed to become proficient is how to use the common SQL commands and concepts. This section completes the primer on MySQL by introducing the most common MySQL commands and concepts.

■ **Note** Rather than regurgitate the reference manual,[9] this section introduces the commands and concepts at a high level. If you decide to use any of the commands or concepts, please refer to the online reference manual for additional details, complete command syntax, and additional examples.

MySQL Commands

This section reviews the most common SQL and MySQL-specific commands that you will need to know to get the most out of your IOT database. While you have already seen some of these in action, this section provides additional information to help you use them.

■ **Note** Case sensitivity of user-supplied variables (for example, last_name versus Last_Name) is not consistent across platforms. For example, case-sensitivity behavior is different on Windows than it is on Mac OS X. MySQL adheres to the platform's case-sensitivity policy. Check the online reference manual for your platform to see how case sensitivity affects user-supplied variables.

---

[9]Now, that's one trick the professional regurgitator might not be able to do. See https://en.wikipedia.org/wiki/Stevie_Starr.

Creating Databases and Tables

The most basic commands you will need to learn and master are the `CREATE DATABASE` and `CREATE TABLE` commands. Recall that database servers such as MySQL allow you to create any number of databases that you can add tables and store data in a logical manner.

To create a database, use `CREATE DATABASE` followed by a name for the database. If you are using the MySQL client, you must use the `USE` command to switch to a specific database. The client focus is the latest database specified either at startup (on the command line) or via the `USE` command. You can override this by referencing the database name first. For example, `SELECT * FROM db1.table1` will execute regardless of the default database set. However, leaving off the database name will cause the mysql client to use the default database. The following shows two commands to create and change the focus of the database:

```
mysql> CREATE DATABASE plant_monitoring;
mysql> USE plant_monitoring;
```

■ **Tip** Recall if you want to see all the databases on the server, use the `SHOW DATABASES` command.

Creating a table requires the, yes, `CREATE TABLE` command. This command has many options allowing you to specify not only the columns and their data types but also additional options such as indexes, foreign keys, and so on. An index can also be created using the `CREATE INDEX` command (see the following code). The following shows how to create a simple table for storing plant sensor data.

```
CREATE TABLE `plant_monitoring`.`plants` (
  `plant_name` char(30) NOT NULL,
  `sensor_value` float DEFAULT NULL,
  `sensor_event` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `sensor_level` char(5) DEFAULT NULL,
  PRIMARY KEY `plant_name` (`plant_name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Notice here that I specified the table name (`plants`) and four columns (`plant_name`, `sensor_value`, `sensor_event`, and `sensor_level`). I used several data types. For `plant_name`, I used a character field with a maximum of 30 characters, a floating-point data type for `sensor_value`, a timestamp value for `sensor_event`, and another character field for `sensor_level` of five characters.

The `TIMESTAMP` data type is of particular use in IOT solutions or any time you want to record the date and time of an event or action. For example, it is often helpful to know when a sensor value is read. By adding a `TIMESTAMP` column to the table, you do not need to calculate, read, or otherwise format a date and time at the sensor or even aggregate node.

Notice also that I specified that the `sensor_name` column be defined as a key, which creates an index. In this case, it is also the primary key. The `PRIMARY KEY` phrase tells the server to ensure there exists one and only one row in the table that matches the value of the column. You can specify several columns to be used in the primary key by repeating the keyword. Note that all primary key columns must not permit nulls (`NOT NULL`).

If you cannot determine a set of columns that uniquely identify a row (and you want such a behavior—some favor tables without this restriction, but a good DBA would not), you can use an artificial data type option for integer fields called `AUTO INCREMENT`. When used on a column (must be the first column), the server automatically increases this value for each row inserted. In this way, it creates a default primary key. For more information about auto increment columns, see the online reference manual.

■ **Tip** Best practices suggest using a primary key on a character field to be suboptimal in some situations such as tables with large values for each column or many unique values. This can make searching and indexing slower. In this case, you could use an auto increment field to artificially add a primary key that is smaller in size (but somewhat more cryptic).

There are far more data types available than those shown in the previous example. You should review the online reference manual for a complete list of data types. See the section "Data Types." If you want to know the layout or "schema" of a table, use the SHOW CREATE TABLE command.

Like databases, you can also get a list of all the tables in the database with the SHOW TABLES command.

Getting Results

The most used basic command you need to know is the command to return the data from the table (also called a *result set* or *rows*). To do this, you use the SELECT statement. This SQL statement is the workhorse for a database system. All queries for data will be executed with this command.[10] As such, we will spend a bit more time looking at the various clauses (parts) that can be used starting with the column list.

The SELECT statement allows you to specify which columns you want to choose from the data. The list appears as the first part of the statement. The second part is the FROM clause, which specifies the table(s) you want to retrieve rows from.

■ **Note** The FROM clause can be used to join tables with the JOIN operator. You will see a simple example of a join in a later section.

The order that you specify the columns determines the order shown in the result set. If you want all of the columns, use an asterisks (*) instead. Listing 5-4 demonstrates three statements that generate the same result sets. That is, the same rows will be displayed in the output of each. In fact, I am using a table with only four rows for simplicity.

Listing 5-4. Example SELECT Statements

```
mysql> SELECT plant_name, sensor_value, sensor_event, sensor_level FROM
plant_monitoring.plants;
+------------------------+--------------+---------------------+--------------+
| plant_name             | sensor_value | sensor_event        | sensor_level |
+------------------------+--------------+---------------------+--------------+
| fern in den            |       0.2319 | 2015-09-23 21:04:35 | NULL         |
| fern on deck           |         0.43 | 2015-09-23 21:11:45 | NULL         |
| flowers in bedroom1    |        0.301 | 2015-09-23 21:11:45 | NULL         |
| weird plant in kitchen |        0.677 | 2015-09-23 21:11:45 | NULL         |
+------------------------+--------------+---------------------+--------------+
4 rows in set (0.00 sec)
```

---

[10]Not including direct, engine-level queries like NoSQL using NDB.

```
mysql> SELECT * FROM plant_monitoring.plants;
+----------------------+--------------+---------------------+--------------+
| plant_name           | sensor_value | sensor_event        | sensor_level |
+----------------------+--------------+---------------------+--------------+
| fern in den          |       0.2319 | 2015-09-23 21:04:35 | NULL         |
| fern on deck         |         0.43 | 2015-09-23 21:11:45 | NULL         |
| flowers in bedroom1  |        0.301 | 2015-09-23 21:11:45 | NULL         |
| weird plant in kitchen |     0.677 | 2015-09-23 21:11:45 | NULL         |
+----------------------+--------------+---------------------+--------------+
4 rows in set (0.00 sec)

mysql> SELECT sensor_value, plant_name, sensor_level, sensor_event FROM
plant_monitoring.plants;
+--------------+----------------------+--------------+---------------------+
| sensor_value | plant_name           | sensor_level | sensor_event        |
+--------------+----------------------+--------------+---------------------+
|       0.2319 | fern in den          | NULL         | 2015-09-23 21:04:35 |
|         0.43 | fern on deck         | NULL         | 2015-09-23 21:11:45 |
|        0.301 | flowers in bedroom1  | NULL         | 2015-09-23 21:11:45 |
|        0.677 | weird plant in kitchen | NULL       | 2015-09-23 21:11:45 |
+--------------+----------------------+--------------+---------------------+
4 rows in set (0.00 sec)
```

Notice that the first two statements result in the same rows as well as the same columns in the same order, but the third statement, while it generates the same rows, displays the columns in a different order.

You can also use functions in the column list to perform calculations and similar operations. One special example is using the COUNT() function to determine the number of rows in the result set, as shown here. See the online reference manual for more examples of functions supplied by MySQL.

```
SELECT COUNT(*) FROM plant_monitoring.plants;
```

The next clause in the SELECT statement is the WHERE clause. This is where you specify the conditions you want to use to restrict the number of rows in the result set. That is, only those rows that match the conditions. The conditions are based on the columns and can be quite complex. That is, you can specify conditions based on calculations, results from a join, and more. But most conditions will be simple equalities or inequalities on one or more columns in order to answer a question. For example, suppose you wanted to see the plants where the sensor value read is less than 0.40. In this case, we issue the following query and receive the results. Notice I specified only two columns: the plant name and the value read from sensor.

```
mysql> SELECT plant_name, sensor_value FROM plant_monitoring.plants WHERE
sensor_value < 0.40;
+----------------------+--------------+
| plant_name           | sensor_value |
+----------------------+--------------+
| fern in den          |       0.2319 |
| flowers in bedroom1  |        0.301 |
+----------------------+--------------+
2 rows in set (0.01 sec)
```

There are additional clauses you can use including the GROUP BY clause, which is used for grouping rows for aggregation or counting, and the ORDER BY clause, which is used to order the result set. Let's take a quick look at each starting with aggregation.

Suppose you wanted to average the sensor values read in the table for each sensor. In this case, we have a table that contains sensor readings over time for a variety of sensors. While the example contains only four rows (and thus may not be statistically informative), the example demonstrates the concept of aggregation quite plainly, as shown in Listing 5-5. Notice what we receive is simply the average of the four sensor values read.

Listing 5-5. GROUP BY Example

```
mysql> SELECT plant_name, sensor_value FROM plant_monitoring.plants WHERE plant_name =
'fern on deck';
+--------------+--------------+
| plant_name   | sensor_value |
+--------------+--------------+
| fern on deck |         0.43 |
| fern on deck |         0.51 |
| fern on deck |        0.477 |
| fern on deck |         0.73 |
+--------------+--------------+
4 rows in set (0.00 sec)

mysql> SELECT plant_name, AVG(sensor_value) as avg_value FROM plant_monitoring.plants
WHERE plant_name = 'fern on deck' GROUP BY plant_name;
+--------------+-------------------+
| plant_name   | avg_value         |
+--------------+-------------------+
| fern on deck | 0.536750003695488 |
+--------------+-------------------+
1 row in set (0.00 sec)
```

Notice I specified the average function, AVG(), in the column list and passed in the name of the column I wanted to average. There are many such functions available in MySQL to perform some powerful calculations. Clearly, this is another example of how much power exists in the database server that would require many more resources on a typical lightweight sensor or aggregator node in the network.

Notice also that I renamed the column with the average with the AS keyword. You can use this to rename any column specified, which changes the name in the result set, as you can see in the listing.

Another use of the GROUP BY clause is counting. In this case, we replaced AVG() with COUNT() and received the number of rows matching the WHERE clause. More specifically, we want to know how many sensor values were stored for each plant.

```
mysql> SELECT plant_name, COUNT(sensor_value) as num_values FROM plant_monitoring.plants
GROUP BY plant_name;
+-----------------------+------------+
| plant_name            | num_values |
+-----------------------+------------+
| fern in den           |          1 |
| fern on deck          |          4 |
| flowers in bedroom1   |          1 |
| weird plant in kitchen |         1 |
+-----------------------+------------+
4 rows in set (0.00 sec)
```

Now let's say we want to see the results of our result set ordered by sensor value. We will use the same query that selected the rows for the fern on the deck, but we order the rows by sensor value in ascending and descending order using the ORDER BY clause. Listing 5-6 shows the results of each option.

Listing 5-6. ORDER BY Examples

```
mysql> SELECT plant_name, sensor_value FROM plant_monitoring.plants WHERE plant_name =
'fern on deck' ORDER BY sensor_value ASC;
+--------------+--------------+
| plant_name   | sensor_value |
+--------------+--------------+
| fern on deck |         0.43 |
| fern on deck |        0.477 |
| fern on deck |         0.51 |
| fern on deck |         0.73 |
+--------------+--------------+
4 rows in set (0.00 sec)

mysql> SELECT plant_name, sensor_value FROM plant_monitoring.plants WHERE plant_name =
'fern on deck' ORDER BY sensor_value DESC;
+--------------+--------------+
| plant_name   | sensor_value |
+--------------+--------------+
| fern on deck |         0.73 |
| fern on deck |         0.51 |
| fern on deck |        0.477 |
| fern on deck |         0.43 |
+--------------+--------------+
4 rows in set (0.00 sec)
```

As I mentioned, there is a lot more to the SELECT statement than shown here, but what we have seen here will get you very far, especially when working with data typical of most small to medium-sized IOT solutions.

Adding Data

Now that you have a database and tables created, you will want to load or insert data into the tables. You can do so using the INSERT INTO statement. Here we specify the table and the data for the row. The following shows a simple example:

```
INSERT INTO plant_monitoring.plants (plant_name, sensor_value) VALUES ('fern in den', 0.2319);
```

In this example, I am inserting data for one of my plants by specifying the name and value. What about the other columns, you wonder? In this case, the other columns include a timestamp column, which will be filled in by the database server. All other columns (just the one) will be set to NULL, which means no value is available, the value is missing, the value is not zero, or the value is empty.[11]

---

[11]https://en.wikipedia.org/wiki/Null_%28SQL%29

Notice I specified the columns before the data for the row. This is necessary whenever you want to insert data for fewer columns than what the table contains. More specifically, leaving the column list off means you must supply data (or NULL) for all columns in the table. Also, the order of the columns listed can be different from the order they are defined in the table. Leaving the column list off will result in the ordering the column data based on how they appear in the table.[12]

You can also insert several rows using the same command by using a comma-separated list of the row values, as shown here:

```
INSERT INTO plant_monitoring.plants (plant_name, sensor_value) VALUES ('flowers in
bedroom1', 0.301), ('weird plant in kitchen', 0.677), ('fern on deck', 0.430);
```

Here I've inserted several rows with the same command. Note that this is just a shorthand mechanism and, except for automatic commits, no different than issuing separate commands.[13]

Changing Data

There are times when you want to change or update data. You may have a case where you need to change the value of one or more columns, replace the values for several rows, or correct formatting or even scale of numerical data. To update data, we use the UPDATE command.

You can update a particular column, update a set of columns, perform calculations one or more columns, and more. I do not normally have much need to change data in an IOT solution, but sometimes in the case of mistakes in sensor-reading code or similar data entry problems, it may be necessary.

What may be more likely is you or your users will want to rename an object in your database. For example, suppose we determine the plant on the deck is not actually a fern but was an exotic flowering plant.[14] In this case, we want to change all rows that have a plant name of "fern on deck" to "flowers on deck." The following command performs the change:

```
UPDATE plant_monitoring.plants SET plant_name = 'flowers on deck' WHERE plant_name =
'fern on deck';
```

Notice the key operator here is the SET operator. This tells the database to assign a new value to the column(s) specified. You can list more than one set operation in the command.

Notice I used a WHERE clause here to restrict the UPDATE to a particular set of rows. This is the same WHERE clause as you saw in the SELECT statement, and it does the same thing; it allows you to specify conditions that restrict the rows affected. If you do not use the WHERE clause, the updates will apply to all rows.

■ **Caution**   Don't forget the WHERE clause! Issuing an UPDATE command without a WHERE clause will affect all rows in the table!

---

[12]If you're a relational database expert like me, ordering concepts like this in database systems and especially SQL makes my skin crawl. So much for the "unordered" concept!

[13]See the online reference manual for additional conditions and differences of the multiple row insert command.

[14]Hey, it happens. While this is fictional, my wife and I discovered a plant we thought was one thing turned out to be another when it started to bloom.

Removing Data

Sometimes you end up with data in a table that needs to be removed. Maybe you used test data and want to get rid of the fake rows, or perhaps you want to compact or purge your tables or want to eliminate rows that no longer apply. To remove rows, use the DELETE FROM command.

Let's look at an example. Suppose you have a plant-monitoring solution under development and you've discovered that one of your sensors or sensor nodes are reading values that are too low, because of a coding, wiring, or calibration error. In this case, we want to remove all rows with a sensor value less than 0.20. The following command does this:

```
DELETE FROM plants WHERE sensor_value < 0.20;
```

■ **Caution**   Don't forget the WHERE clause! Issuing a DELETE FROM command without a WHERE clause will permanently delete all rows in the table!

Notice I used a WHERE clause here. That is, a conditional statement to limit the rows acted upon. You can use whatever columns or conditions you want; just be sure you have the correct ones! I like to use the same WHERE clause in a SELECT statement first. For example, I would issue the following first to check that I am about to delete the rows I want and only those rows. Notice it is the same WHERE clause.

```
SELECT * FROM plants WHERE sensor_value < 0.20;
```

MySQL Concepts

Besides the commands shown earlier, there are additional concepts that you may want to consider using. While it is true each has one SQL command, I list them here as a concept because they are more complex than a simple object creation or data retrieval.

Indexes

Tables are created without the use of any ordering. That is, tables are unordered. While it is true MySQL will return the data in the same order each time, there is no implied (or reliable) ordering unless you create an index. The ordering I am referring to here is not like you think when sorting (that's possible with the ORDER BY clause in the SELECT statement).

Rather, indexes are mappings that the server uses to read the data when queries are executed. For example, if you had no index on a table and wanted to select all rows with a value greater than a certain value for a column, the server will have to read all rows to find all the matches. However, if we added an index on that column, the server would have to read only those rows that match the criteria.

I should note that there are several forms of indexes. What I am referring to here is a clustered index where the value for column in the index is stored in the index, allowing the server to read the index only and not the rows to do the test for the criteria.

To create an index, you can either specify the index in the CREATE TABLE statement or issue a CREATE INDEX command. The following shows a simple example:

```
CREATE INDEX plant_name ON plants (plant_name);
```

This command adds an index on the plant_name column. Observe how this affects the table.

```
CREATE TABLE `plants` (
  `plant_name` char(30) DEFAULT NULL,
  `sensor_value` float DEFAULT NULL,
  `sensor_event` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `sensor_level` char(5) DEFAULT NULL,
  KEY `plant_name` (`plant_name`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Indexes created like this do not affect the uniqueness of the rows in the table, in other words, making sure there exists one and only one row that can be accessed by a specific value of a specific column (or columns). What I am referring to is the concept of a primary key (or primary index), which is a special option used in the creation of the table as described earlier.

Views

Views are logical mappings of results of one or more tables. They can be referenced as if they were tables in queries, making them a powerful tool for creating subsets of data to work with. You create a view with CREATE VIEW and give it a name similar to a table. The following shows a simple example where we create a test view to read values from a table. In this case, we limit the size of the view (number of rows), but you could use a wide variety of conditions for your views, including combining data from different tables.

```
CREATE VIEW test_plants AS SELECT * FROM plants LIMIT 5;
```

Views are not normally encountered in small or medium-sized database solutions, but I include them to make you aware of them in case you decide to do additional analysis and want to organize the data into smaller groups for easier reading.

Triggers

Another advanced concept (and associated SQL command) is the use of an event-driven mechanism that is "triggered" when data is changed. That is, you can create a short set of SQL commands (a procedure) that will execute when data is inserted or changed.

There are several events or conditions under which the trigger will execute. You can set up a trigger either before or after an update, insert, or delete action. A trigger is associated with a single table and has as its body a special construct that allows you to act on the rows affected. The following shows a simple example:

```
DELIMITER //
CREATE TRIGGER set_level BEFORE INSERT ON plants FOR EACH ROW
BEGIN
  IF NEW.sensor_value < 0.40 THEN
    SET NEW.sensor_level = 'LOW';
  ELSEIF NEW.sensor_value < 0.70 THEN
    SET NEW.sensor_level = 'OK';
  ELSE
    SET NEW.sensor_level = 'HIGH';
  END IF;
END //
DELIMITER ;
```

This trigger will execute before each insert into the table. As you can see in the compound statement (BEGIN...END), we set a column called sensor_level to LOW, OK, or HIGH depending on the value of the sensor_value. To see this in action, consider the following command. The FOR EACH ROW syntax allows the trigger to act on all rows in the transaction.

```
INSERT INTO plants (plant_name, sensor_value) VALUES ('plant1', 0.5544);
```

Since the value we supplied is less than the middle value (0.70), we expect the trigger to fill in the sensor_level column for us. The following shows this indeed is what happened when the trigger fired:

```
+------------+--------------+---------------------+--------------+
| plant_name | sensor_value | sensor_event        | sensor_level |
+------------+--------------+---------------------+--------------+
| plant1     |       0.5544 | 2015-09-23 20:00:15 | OK           |
+------------+--------------+---------------------+--------------+
1 row in set (0.00 sec)
```

This demonstrates an interesting and powerful way you can create derived columns with the power of the database server and save the processing power of your sensor or aggregator nodes. I encourage you to consider this and similar powerful concepts for leveraging the power of the database server.

Simple Joins

One of the most powerful concepts of database systems is the ability to make relationships (hence the name relational) among the data. That is, data in one table can reference data in another (or several tables). The most simplistic form of this is called a *master-detail* relationship where a row in one table references or is related to one or more rows in another.

A common (and classic) example of a master-detail relationship is from an order-tracking system where we have one table containing the data for an order and another table containing the line items for the order. Thus, we store the order information such as customer number and shipping information once and combine or "join" the tables when we retrieve the order proper.

Let's look at an example from the sample database named world. You can find this database on the MySQL web site (http://dev.mysql.com/doc/index-other.html). Feel free to download it and any other sample database. They all demonstrate various designs of database systems. You will also find it handy to practice querying the data as it contains more than a few, simple rows.

■ **Note** If you want to run the following examples, you need to install the world database as described in the documentation for the example (http://dev.mysql.com/doc/world-setup/en/world-setup-installation.html).

Listing 5-7 shows an example of a simple join. There is a lot going on here, so take a moment to examine the parts of the SELECT statement, especially how I specified the JOIN clause. You can ignore the LIMIT option because that simply limits the number of rows in the result set.

Listing 5-7. Simple JOIN Example

```
mysql> SELECT Name, Continent, Language FROM Country JOIN CountryLanguage ON Country.Code =
CountryLanguage.CountryCode LIMIT 10;
+-------------+---------------+------------+
| Name        | Continent     | Language   |
+-------------+---------------+------------+
| Aruba       | North America | Dutch      |
| Aruba       | North America | English    |
| Aruba       | North America | Papiamento |
| Aruba       | North America | Spanish    |
| Afghanistan | Asia          | Balochi    |
| Afghanistan | Asia          | Dari       |
| Afghanistan | Asia          | Pashto     |
| Afghanistan | Asia          | Turkmenian |
| Afghanistan | Asia          | Uzbek      |
| Angola      | Africa        | Ambo       |
+-------------+---------------+------------+
10 rows in set (0.00 sec)
```

Here I used a JOIN clause that takes two tables specified such that the first table is joined to the second table using a specific column and its values (the ON specifies the match). What the database server does is read each row from the tables and returns only those rows where the value in the columns specified a match. Any rows in one table that are not in the other are not returned.

■ **Tip** But you can retrieve those rows with different joins. See the online reference manual on inner and outer joins for more details.

Notice also that I included only a few columns. In this case, I specified the country name and continent from the Country table and the language column from the CountryLanguage table. If the column names were not unique (the same column appears in each table), I would have to specify them by table name such as Country.Name. In fact, it is considered good practice to always qualify the columns in this manner.

There is one interesting anomaly in this example that I feel important to point out. In fact, some would consider it a design flaw. Notice in the JOIN clause I specified the table and column for each table. This is normal and correct, but notice the column name does not match in both tables. While this really doesn't matter and creates only a bit of extra typing, some DBAs would consider this erroneous and would have a desire to make the common column name the same in both tables.

Another use for a join is to retrieve common, archival, or lookup data. For example, suppose you had a table that stored details about things that do not change (or rarely change) such as cities associated with ZIP codes or names associated with identification numbers (e.g., SSN). You could store this information in a separate table and join the data on a common column (and values) whenever you needed. In this case, that common column can be used as a foreign key, which is another advanced concept.

Foreign keys are used to maintain data integrity (that is, if you have data in one table that relates to another table but the relationship needs to be consistent). For example, if you wanted to make sure when you delete the master row that all of the detail rows are also deleted, you could declare a foreign key in the master table to a column (or columns) to the detail table. See the online reference manual for more information about foreign keys.

This discussion on joins touches only the very basics. Indeed, joins are arguably one of the most difficult and often confused areas in database systems. If you find you want to use joins to combine several tables or extend data so that data is provided from several tables (outer joins), you should spend some time with an in-depth study of database concepts such as Clare Churcher's book *Beginning Database Design* (Apress, 2012).

Additional Advanced Concepts

There are more concepts and commands available in MySQL, but two that may be of interest are `PROCEDURE` and `FUNCTION`, sometimes called *routines*. I introduce these concepts here so that if you want to explore them, you understand how they are used at a high level.

Suppose you need to run several commands to change data. That is, you need to do some complex changes based on calculations. For these types of operations, MySQL provides the concept of a stored procedure. The stored procedure allows you to execute a compound statement (a series of SQL commands) whenever the procedure is called. Stored procedures are sometimes considered an advanced technique used mainly for periodic maintenance, but they can be handy in even the more simplistic situations.

For example, suppose you want to develop your IOT solution, but since you are developing it, you need to periodically start over and want to clear out all the data first. If you had only one table, a stored procedure would not help much, but suppose you have several tables spread over several databases (not unusual for larger IOT solutions). In this case, a stored procedure may be helpful.

■ **Tip** When entering commands with compound statements in the MySQL client, you need to change the delimiter (the semicolon) temporarily so that the semicolon at the end of the line does not terminate the command entry. For example, use `DELIMITER //` before writing the command with a compound statement, use `//` to end the command, and change the delimiter back with `DELIMITER ;`. This is only when using the client.

Since stored procedures can be quite complicated, if you decide to use them, read the "CREATE PROCEDURE and CREATE FUNCTION Syntax" section of the online reference manual before trying to develop your own. There is more to creating stored procedures than described in this section.

Now suppose you want to execute a compound statement and return a result—you want to use it as a function. You can use functions to fill in data by performing calculations, data transformation, or simple translations. Functions therefore can be used to provide values to populate column values, provide aggregation, provide date operations, and more.

You have already seen a couple of functions (`COUNT`, `AVG`). These are considered built-in functions, and there is an entire section devoted to them in the online reference manual. However, you can also create your own functions. For example, you may want to create a function to perform some data normalization on your data. More specifically, suppose you have a sensor that produces a value in a specific range, but depending on that value and another value from a different sensor or lookup table, you want to add, subtract, average, and so on, the value to correct it. You could write a function to do this and call it in a trigger to populate the value for a calculation column.

■ **Tip** Use a new column for calculated values so that you preserve the original value.

WHAT ABOUT CHANGING OBJECTS?

You may be wondering what you do when you need to modify a table, procedure, trigger, and so on. Rest easy, you do not have to start over from scratch! MySQL provides an ALTER command for each object. That is, there is an ALTER TABLE, ALTER PROCEDURE, and so on. See the online reference manual section entitled "Data Definition Statements" for more information about each ALTER command.

Planning Database Storage for IOT Data

Now that you know how to get, install, and use MySQL, it's time to focus on how to apply what you learned in previous chapters and set up a database for storing IOT data. Recall IOT data can be any form of sensor data, personal information, codes, dates of events, identification of devices, and so on.

I present this topic by way of example. More specifically, I feel it is best to demonstrate database design rather than dictate practice and policy by rhetoric. I think the examples show many of the concepts and constructs you are likely to encounter when designing your own database(s) for your IOT solutions. Let's dive in with a complete design for a plant-monitoring solution.

The following may seem familiar since I used some primitives of these tables in previous sections. However, this section contains a fully developed database design. This is just one possible design I could have used. I challenge you to consider alternatives should you consider implementing your own plant-monitoring solution.

CORRECT DATABASE DESIGN: AM I DOING THIS RIGHT?

There isn't really any wrong way to design your database. While some DBAs would cringe at such a claim, so long as you can achieve all your goals with reasonable performance and no loss of data, you should consider your design feasible. After all, even the most complex and professionally design databases go through routine and evolutional changes. That is, you don't have to get it right on the first go. You can always adapt your database to your growing and maturing IOT solutions needs.

Example 1: Plant-Monitoring System

Let's explore an IOT solution that monitors ambient temperature and soil moisture for plants. In this case, the solution is designed to support any number of plants (the target data object). A key component of this IOT solution is that all the data collected is stored where the sensors employed are read about once every hour.

In the following example, I show you four basic steps that I like to use when designing a database. You may find other philosophies that have more steps with more rigid processes (and that's great), but for the enthusiasts and hobbyists, I recommend using these simplified steps. That doesn't mean you cannot design any other way, just that this method should work for most. Indeed, if you have experience in designing databases, you should see parallels with your own methodology.

Step 1: Describe the Data

The first thing you should do when designing a database is to describe the data as completely as you can. You should describe the data in English terms perhaps even writing it out on a piece of paper. Doing so helps you define what the data looks like conceptually so that you can determine how many objects you want to store, what they are composed of, and how to organize them.

The plant-monitoring system should store information about plants. Specifically, we want to store information that tells us when plants need watering and how often. It is also important to know the name of the plant, where it is located, and whether it is inside or outside. Thus, the data we need to store consists of the plant name, location, whether they are inside or outside, soil moisture sensor value, temperature sensor value, and time of the sensor reading. We also determine that we want to quantify the value of the soil moisture to make it easy to write an application to detect when a plant needs watering. Indeed, we could even add an automatic watering feature in the future!

Since the plant name is always the same, we don't need to store that information more than once. Thus, we will create one table to store the information about the plant we are monitoring and create another table to store the sensor readings. This way, if you need to change the name of a plant (like we saw previously) or you want to change its location, you need to change it in only one place. Since we will store the sensor readings in a different table, we must choose a column to be used to join the tables. Since we don't have any reasonable numeric value assigned to each plant, we can use the auto increment feature to add a unique key (in this case the primary key).

Step 2: Design the Database Objects

Now let's see how these tables would look like. We name the master table (the one that stores the plant information) `plants` and the detail table (sensor readings) `readings`. We place each of these tables in the database named `plant_monitoring`. Listing 5-8 shows the layout or schema of each of the tables in the database.

Listing 5-8. Plant-Monitoring Schema

```
-- A database for storing plant soil moisture and ambient temperature

CREATE DATABASE plant_monitoring;
USE plant_monitoring;

-- This table stores information about a plant.
CREATE TABLE `plants` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` char(50) DEFAULT NULL,
  `location` char(30) DEFAULT NULL,
  `climate` enum ('inside','outside') DEFAULT 'inside',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- This table stores values read from sensors. The sensors are read
-- periodically by a sensor node and the values stored as a single row
-- with the date and time added by the database server.
```

```
CREATE TABLE `readings` (
  `id` int(11) NOT NULL,
  `moisture` float DEFAULT NULL,
  `temperature` float DEFAULT NULL,
  `event_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `soil_status` enum ('DRY', 'OK', 'WET') DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DELIMITER //
CREATE TRIGGER set_status BEFORE INSERT ON readings FOR EACH ROW
BEGIN
  IF NEW.moisture < 250 THEN
    SET NEW.soil_status = 1;
  ELSEIF NEW.moisture < 400 THEN
    SET NEW.soil_status = 2;
  ELSE
    SET NEW.soil_status = 3;
  END IF;
END //
DELIMITER ;
```

Notice the trigger defined on the readings table. Recall from a previous section that a trigger can be used to provide data for calculated columns, which is what we want to store in the soil_status column. In this case, we set the low threshold for dry soil to 250 and wet for anything over 400. Thus, between 250 and 399, the plant soil is considered normal (Ok).

Also, notice the comments included in the SQL code. The double dash is a comment that the MySQL client will ignore. While this example database is rather simple, it does not hurt to add short descriptions of your database objects (tables, and so on) so that you can remember what each does should you forget or as someone else to work with the database or your IOT solution (like for a bespoke application).

Notice also the location column in the plants table. This is an example of using enumerated values so that you don't have to repeatedly enter the string with each plant you add to the table. Values start at 1, so you can specify 1 for inside and 2 for outside, as shown here:

```
mysql> INSERT INTO plants VALUES(NULL, 'Jerusalem Cherry', 'deck', 2);
Query OK, 1 row affected (0.01 sec)

mysql> select * from plants;
+----+--------------------+----------+---------+
| id | name               | location | climate |
+----+--------------------+----------+---------+
|  1 | Jerusalem Cherry   | deck     | outside |
+----+--------------------+----------+---------+
1 row in set (0.00 sec)
```

■ **Note**   I chose as descriptive names as I could for the columns. I also threw in some that are a bit fuzzy. Can you tell which ones could use a better name? Hint: what does *climate* mean to you? Clearly, choosing valid, meaningful names for columns is a challenge and a bit of an art form.

Finally, notice that while I added a primary key, in this case the AUTO_INCRMENT data type to the plants table, I did not add one to the readings table. This is so that we can store any number of rows in the readings table and that it is entirely possible that the data will not be unique. More precisely, the values read from the sensors may be identical from two or more readings. Thus, I left the uniqueness factor off of the definition of the readings table.

I should also note a consequence of using the AUTO_INCREMENT data type to uniquely identify rows. While conceptually there cannot be more than one plant named the same in the same location in the plants table (even if there were three ferns on the deck, most likely you'd name them differently), the fact that AUTO_INCREMENT is an artificial uniqueness mechanism means you could very well enter the same data twice resulting in different AUTO_INCREMENT values. So, a bit of caution is prudent when working with AUTO_INCREMENT.

There are two handy tools you can use when designing tables. First, you can use the SHOW CREATE TABLE command to see the actual SQL command to re-create the table. In fact, the SHOW CREATE can be used for any object such as SHOW CREATE TRIGGER. Second, you can use the EXPLAIN command, as shown in Listing 5-9.

Listing 5-9. Using EXPLAIN

```
mysql> explain plants;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| name       | char(50) | YES  |     | NULL    |                |
| location   | char(30) | YES  |     | NULL    |                |
| plant_type | char(30) | YES  |     | NULL    |                |
+------------+----------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> explain readings;
+-------------+-------------+------+-----+-------------------+-----------------------------+
| Field       | Type        | Null | Key | Default           | Extra                       |
+-------------+-------------+------+-----+-------------------+-----------------------------+
| id          | int(11)     | NO   |     | NULL              |                             |
| moisture    | float       | YES  |     | NULL              |                             |
| temperature | float       | YES  |     | NULL              |                             |
| event_time  | timestamp   | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| soil_status | enum('DRY', | YES  |     | NULL              |                             |
|             |  'OK','WET')|      |     |                   |                             |
+-------------+-------------+------+-----+-------------------+-----------------------------+
5 rows in set (0.00 sec)
```

Here you see a result set that shows all the columns and their data types and options for each table in the plant monitoring database. Notice the extra information we get for the timestamp field. Here it tells us the timestamp will be updated when the row is updated, but it also applies to a new row.

But wait! How does each sensor know what the ID is for each plant? Well, this is a bit of a cart-and-horse situation. If you consider each sensor may be read and the data sent via a small microcontroller board, then the code (sketch in Arduino lingo) must know the ID from the plants table. Since we used an AUTO_INCREMENT

column in the table, we must first insert the data about the plant and then query the table for its ID. The following shows an example:

```
mysql> INSERT INTO plant_monitoring.plants VALUES (NULL, 'fern', 'beside picnic table', 2);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                9 |
+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM plant_monitoring.plants WHERE id = 9;
+----+------+---------------------+---------+
| id | name | location            | climate |
+----+------+---------------------+---------+
|  9 | fern | beside picnic table | outside |
+----+------+---------------------+---------+
1 row in set (0.00 sec)
```

But notice I did not have to actually query the `plants` table. Instead, I used the `LAST_INSERT_ID()` function, which returns the value of the last `AUTO_INCREMENT` value generated. Cool!

Once we have this information, we can program the sensor reader to send this information to the database to populate the readings table and therefore make the relationship to the plants table. That means when we query for all the sensor readings for a particular plant, the join will return the correct information. Thus, if I wanted to store data about the plant with `id = 9`, I would use an INSERT statement like the following:

```
INSERT INTO plant_monitoring.readings VALUES (9, 233, 38.4, NULL, NULL);
```

DATABASE DESIGN FIRST? WHAT ABOUT THE APPLICATION?

I once worked for an organization that felt the application and user interface were more important than the database design. I guess that's why they eventually hired more DBAs than developers. They had to reengineer the database several times during the development of the application, which caused major delays in deliveries and some interesting and serious bugs.

Always design your database including the questions you want to ask after you've defined the high-level requirements but before you implement the source code. This way, your application will be based on a fully functional database system with well-defined queries rather than struggling to make queries fit some code implementation.

Now let's see how to get information out of the database.

Step 3: Design the Queries or Questions for the Database

Whenever designing a database, I make it a point, if not a required milestone, to determine what queries I want to run on the data (in other words, the questions I want to ask and the answers I expect). These questions are those that would be asked of an application designed to present the information to users. Think of it this way: if you had a plant-monitoring system, what features or results do you expect to see in the user interface? Defining these before you design your application will make the application development proceed much more smoothly.

Queries we may want to perform on a plant-monitoring solution include but are not limited to the following. As you will see, some of these are simple and easy, while others may take a more complex SELECT statement to achieve. I will demonstrate some of these in the following sections, leaving the others for your exploration.

- Which plants need watering?

- Which plants are experiencing the highest temperatures?

- What is the average soil moisture for each plant for each day?

- What is the temperature range that a particular plant endures during the daylight hours?

- How many plants are being monitored?

- How many plants are outside?

- What is the average temperature of the inside or outside plants?

The following sections walk you through how to create the queries for three of these questions. I present the queries first without validation and later will walk through the same logical process showing you how to test each query with known test data. However, for some of the more complex queries, I show you how they work in context so that you can see how the parts fit together.

Example 1: A Simple Query

Let's begin with a simple query. This one involves doing some aggregation. Recall one of the questions we want to ask is, "What is the average temperature of the inside or outside plants?" In this case, let's make it a bit easier and query only for the plants that are outside. As you will see, it is trivial to change the query to find the inside plants.

Like all good query design, we begin by breaking down the problem. For this query, we need to know the average temperature for each plant. Recall we need to use a GROUP BY clause with the AVG() function. But what if we want to restrict the rows to only the readings taken on a given day such as today? That is, our application may provide a periodic status of the average temperature for each plant for the current day. We could query the data over a range of times, but that is more of an analysis operation—not that you couldn't do that. Just to keep it simple, we'll use today's readings.

So, how would you query for such information? Clearly, we need to use some form of date check. Fortunately, MySQL provides many such functions. For this, we will use the DATE() and CURRENT_DATE() functions. These perform the operation of determining the samples that were taken today using the event_time TIMESTAMP column. One of the really cool and powerful features of the database is if we use these functions, we can create queries, views, functions, and so on, that automatically determine which rows were recorded today. Thus, we don't need to read, store, and transmit the date when doing queries using these functions. Cool!

The following WHERE clause demonstrates how to do this. Notice we pass in the event_time column to the function, which then determines the date. We compare this to the current date and thus find only those rows where the sensor reading was taken today.

```
WHERE DATE(event_time) = CURRENT_DATE()
```

Now we just need to do the average. We've already seen an example earlier, so the following SQL command should look familiar—at least in concept:

```
SELECT id, AVG(temperature) as avg_temp FROM readings WHERE DATE(event_time) =
CURRENT_DATE() GROUP BY id;
```

This will give us the average temperature and the ID for each plant for those readings taken today. All that is left is to join this information with the information in the plants table to get the name, location, and average temperature. The following SQL statement shows how to do this:

```
SELECT name, location, AVG(temperature) as avg_temp
FROM plants JOIN readings ON plants.id = readings.id
WHERE DATE(event_time) = CURRENT_DATE()
GROUP BY plants.id;
```

Wow, there is a lot going on here for such a simple query! I placed each clause on a separate line to make it easier to read. Let's review each.

First, we see the columns selected are the plant name and location as well as the average temperature. Recall the GROUP BY clause is what controls which rows are supplied to the function. In this case, we group by the plant ID. Notice the FROM clause performs a join between the plants table and the readings table. This is so that we can get the plant information from the plants table but perform our calculations (average temperature) on the data from the readings table that matches the plants table. Thus, a simple join! Finally, we use the WHERE clause described earlier to restrict the data to only those samples taken today.

But we're not done. We haven't answered the question in its entirety. That is, we also want to know the average temperature for the outside plants only. All we need to do then is add that condition to the WHERE clause as follows:

```
SELECT name, location, AVG(temperature) as avg_temp
FROM plants JOIN readings ON plants.id = readings.id
WHERE DATE(event_time) = CURRENT_DATE() AND plants.climate = 2
GROUP BY plants.id;
```

Now, we're done! To query for the inside plants, just change the climate value to 1. Recall this is the value of the enumerated data type for the climate column.

Now let's look at a much more complex query. At first glance, this will see simple but as you shall see has many layers.

■ **Note** What follows is just one example of how to form the query. There are several others, some more optimal, but I want to demonstrate the logical thought process you could take to solve the problem. Feel free to experiment and improve the following example.

Example 2: A Complex Query

Let's now look at a more complex query. Or rather one that is simple in its formation but not trivial in implementation. In this case, consider the query, "Which plants need watering?" For this, we need to know which of the plants have a moisture value that is below our threshold of DRY/OK/WET. Since we use a calculated column, we do not have to see the actual sensor value. That is, we can query the readings table and discover which are marked DRY.

You may think that we just need to query the readings table for any value of DRY for the sensor readings for today. This will get you close, but it likely result in a number of rows, but what if you or an automatic plant-watering system waters the plant or what if it rains? You could have only a few readings of DRY but more readings of OK or WET. In this case, the plant may not need watering at all.

You may also consider selecting the most recent sensor reading for each plant. Indeed, this is what most people would do. But that won't cover situations where the sensors are reading values on the borderline or produce spurious readings. For example, the hobbyist-level soil moisture sensors (in other words, the affordable ones) are not 100 percent accurate and can produce slightly inconsistent readings. They are perfectly fine for a "broad" reading so long as you understand you should review several readings either averaging or considering the percentage of readings of one range versus another.

What we need is a way to determine those plant readings of DRY that are more frequent than the other values, which requires a bit of mathematics. And you thought this was simple. Well, it is if you break it down into smaller parts, which is why I chose this example. Let's break this down into parts. We begin with the most basic statement of what we want.

More specifically, we want to know the current day's soil status for all plants. We will count the values of soil_status and determine the percentage of the occurrence based on the total readings for the day. We can use that information later to determine which plants need watering.

To make this work, we're going to use a concept in MySQL called a *view*. Recall a view is a logical representation of a result set and can be treated like a table in other SELECT statements. The following shows the view to retrieve the current day's soil status:

```
CREATE VIEW soil_status_today AS
  SELECT id, soil_status, count(soil_status) as num_events FROM plant_monitoring.readings
  WHERE DATE(event_time) = CURRENT_DATE() GROUP BY id, soil_status;
```

Notice also I used a GROUP BY clause to aggregate the values counting how many were each status value. Let's see an example result. Notice I query the view just like a table.

```
mysql> SELECT * FROM plant_monitoring.soil_status_today;
+----+-------------+------------+
| id | soil_status | num_events |
+----+-------------+------------+
|  1 | DRY         |         10 |
|  2 | OK          |         10 |
|  3 | DRY         |          4 |
|  3 | OK          |          4 |
|  3 | WET         |          2 |
|  4 | OK          |          6 |
|  4 | WET         |          4 |
|  5 | OK          |         10 |
+----+-------------+------------+
8 rows in set (0.01 sec)
```

So, this tells us that, for today, plant IDs 1 and 3 are dry. But we're not done! Consider there are multiple samples taken during the day. Some plants may be on the threshold where they are a little dry but not so much that the sensor readings are consistent. So, we want plants that are consistently dry where there are more DRY events read than OK or WET. We could use another view to get this information, but let's see how we can do this with a stored function. The following creates a function that returns the maximum number of samples collected for today for a given plant:

```
DELiMITER //
CREATE FUNCTION plant_monitoring.max_samples_today (in_id int)
RETURNS int DETERMINISTIC READS SQL DATA
BEGIN
  DECLARE num_samples int;
  SELECT COUNT(*) into num_samples FROM plant_monitoring.readings
  WHERE DATE(event_time) = CURRENT_DATE() AND readings.id = in_id;
  RETURN num_samples;
END //
DELIMITER ;
```

Let's see how this function works. Let's create a query using the view and calculate the percentage of occurrence for each value for each plant. The following SELECT statement accomplishes this with a bit of mathematics. I include the rows to show you that it works.

```
mysql> SELECT *, max_samples_today(id) as max_samples, (num_events/max_samples_today(id)) as
percent_occurrence FROM plant_monitoring.soil_status_today;
+----+-------------+------------+-------------+--------------------+
| id | soil_status | num_events | max_samples | percent_occurrence |
+----+-------------+------------+-------------+--------------------+
|  1 | DRY         |         10 |          10 |             1.0000 |
|  2 | OK          |         10 |          10 |             1.0000 |
|  3 | DRY         |          4 |          10 |             0.4000 |
|  3 | OK          |          4 |          10 |             0.4000 |
|  3 | WET         |          2 |          10 |             0.2000 |
|  4 | OK          |          6 |          10 |             0.6000 |
|  4 | WET         |          4 |          10 |             0.4000 |
|  5 | OK          |         10 |          10 |             1.0000 |
+----+-------------+------------+-------------+--------------------+
8 rows in set (0.01 sec)
```

In this case, I have exactly ten sensor readings for each plant for today. This is because my test data (which I will show you in a later section) is fixed. When I run this on my live plant-monitoring solution in my home, my sensor readings average about one per hour with 18 to 24 readings per day.

Notice I added the function call to get the maximum number of samples (readings) and then another column calculating the percentage of the total occurrences. But this is a lot of work. We can do it a bit easier now that we know how to do mathematical operations in the query.

All we need to do now is add a check for the percentage, say more than 50 percent, and restrict the rows to those with a soil_status of DRY. Thus, we take the previous query and add a few more conditions.

We will also limit the columns in the result to just the id. The following query shows an example SELECT statement to determine which readings indicate which plant or plants need watering today:

```
mysql> SELECT id FROM soil_status_today WHERE ((num_events/max_samples_today(id)) > 0.50)
AND soil_status = 1;
+----+
| id |
+----+
|  1 |
+----+
1 row in set (0.00 sec)
```

We're almost there. Once again, we will use the previous query and join against the plant table to determine which plants need monitoring.

```
SELECT name, location FROM plants JOIN (SELECT id FROM soil_status_today WHERE ((num_events/
max_samples_today(id)) > 0.50) AND soil_status = 1) as sub_query on plants.id = sub_query.id;
```

And now we know the plant that needs watering today. Notice we simply used our logical breakdown of how to find the data starting from the most basic and moving outward (as in narrowing our results). As you can see, this query wasn't nearly that difficult, and since we created a function and a view to help us, queries similar to this one will be much easier to write.

Step 4: Testing the Database

Now that we have the database designed, implemented, and the queries decided upon (or at least all of the ones we can think of), we are ready to start building our sensor network and code the application, right? Nope. We need to test the queries not only to make sure they work (that is, have the correct syntax) but also to make sure we are getting the results we expect. This requires having a known data set to work with. It will do you little good if you find out after your application has gone viral that one of the queries doesn't return the correct data.

At this point, the data may not be completely accurate nor does it require actual, live data. You can use made-up data so long as you make data that is representative of the range of values for each column in the table. That is, make sure you know what the min and max values are that your sensors can read. Listing 5-10 shows some sample data I created to test the queries for the plant-monitoring solution.

Listing 5-10. Sample Data

```
INSERT INTO plant_monitoring.plants VALUES (NULL, 'Jerusalem Cherry', 'deck', 2);
INSERT INTO plant_monitoring.plants VALUES (NULL, 'Moses in the Cradle', 'patio', 2);
INSERT INTO plant_monitoring.plants VALUES (NULL, 'Peace Lilly', 'porch', 1);
INSERT INTO plant_monitoring.plants VALUES (NULL, 'Thanksgiving Cactus', 'porch', 1);
INSERT INTO plant_monitoring.plants VALUES (NULL, 'African Violet', 'porch', 1);

INSERT INTO plant_monitoring.readings VALUES (1, 235, 39.9, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (1, 235, 38.7, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (1, 230, 38.8, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (1, 230, 39.1, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (1, 215, 39.2, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (1, 215, 39.5, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (1, 225, 39.2, NULL, NULL);
```

```
INSERT INTO plant_monitoring.readings VALUES (1, 220, 38.9, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (1, 222, 38.5, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (1, 218, 37.1, NULL, NULL);

INSERT INTO plant_monitoring.readings VALUES (2, 355, 38.1, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (2, 350, 38.6, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (2, 366, 38.7, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (2, 378, 38.8, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (2, 361, 38.7, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (2, 348, 37.5, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (2, 343, 39.1, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (2, 342, 38.8, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (2, 358, 36.9, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (2, 377, 36.1, NULL, NULL);

INSERT INTO plant_monitoring.readings VALUES (3, 155, 33.6, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (3, 150, 33.7, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (3, 166, 33.6, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (3, 278, 32.3, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (3, 261, 31.2, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (3, 248, 32.5, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (3, 313, 33.6, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (3, 342, 32.8, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (3, 458, 31.9, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (3, 470, 33.4, NULL, NULL);

INSERT INTO plant_monitoring.readings VALUES (4, 333, 33.1, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (4, 345, 33.6, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (4, 360, 34.4, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (4, 380, 34.2, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (4, 395, 33.7, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (4, 385, 33.4, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (4, 425, 32.3, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (4, 420, 31.1, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (4, 422, 33.8, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (4, 418, 32.5, NULL, NULL);

INSERT INTO plant_monitoring.readings VALUES (5, 335, 39.9, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (5, 335, 38.7, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (5, 330, 38.8, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (5, 330, 39.1, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (5, 315, 39.2, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (5, 315, 39.5, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (5, 325, 39.2, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (5, 320, 38.9, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (5, 322, 38.5, NULL, NULL);
INSERT INTO plant_monitoring.readings VALUES (5, 318, 37.1, NULL, NULL);
```

Notice I used NULL for the last two columns. This is habit of mine from my DBA days where I always supply a value for every field in the table. You do not have to do it that way. Indeed, it is also correct to specify the SQL commands with just the values you know, but remember to use the column list, as shown here:

INSERT INTO plant_monitoring.readings (id, moisture, temperature) VALUES (7,418,32.5);

As a simple test, let's do a simple join to find all the readings for a particular plant. For example, we ask the question, "What are the moisture readings for the plants on the deck?" The following query returns the results:

```
mysql> SELECT name, location, moisture FROM plants JOIN readings ON plants.id = readings.id
WHERE location LIKE '%deck%';
+------+----------+----------+
| name | location | moisture |
+------+----------+----------+
| fern | deck     |      235 |
| fern | deck     |      235 |
| fern | deck     |      230 |
| fern | deck     |      230 |
| fern | deck     |      215 |
| fern | deck     |      215 |
| fern | deck     |      225 |
| fern | deck     |      220 |
| fern | deck     |      222 |
| fern | deck     |      218 |
+------+----------+----------+
10 rows in set (0.00 sec)
```

Notice I used a LIKE function in the WHERE clause. I used it because I wasn't sure if there were more than one plant on the deck. For example, there could have been a plant whose location was, "deck by table" or "on deck under tree." Using the LIKE with a wildcard, %, on each side returns all rows that have "deck" in the location column value. Cool! Take some time to look at the sample data in Listing 5-10 to ensure you are seeing the correct data.

Now that we have some sample data, let's see the results of the example queries. I walk through the results in the same order that I explained the previous query.

Testing Example 1

Recall for this query we simply want the average temperature for the plants that are outside. When I test a query, I like to break it down into its most simplistic parts—the same way I did to develop it. This way, I can verify I am getting the correct results for each part. Let's see that query again.

```
SELECT name, location, AVG(temperature) as avg_temp
FROM plants JOIN readings ON plants.id = readings.id
WHERE DATE(event_time) = CURRENT_DATE() AND plants.climate = 2
GROUP BY plants.id;
```

Let's begin with the most basic data—the name and location of the plants that live outside.

■ **Note** I will enter the query by parts so you can read the results better. As you can see, it is much easier to read in the MySQL client this way, and the client will "wait" until you type the semicolon to execute the query.

```
mysql> SELECT * FROM plants
    -> WHERE climate = 2;
+----+--------------------+----------+---------+
| id | name               | location | climate |
+----+--------------------+----------+---------+
|  1 | Jerusalem Cherry   | deck     | outside |
|  2 | Moses in the Cradle | patio   | outside |
+----+--------------------+----------+---------+
2 rows in set (0.00 sec)
```

So, we see there are two plants outside, one on the deck and another on the patio. Now, what is the average temperature of each plant's readings for today?

```
mysql> SELECT id, AVG(temperature)
    -> FROM readings
    -> WHERE DATE(event_time) = CURRENT_DATE() GROUP BY id;
+----+--------------------+
| id | AVG(temperature)   |
+----+--------------------+
|  1 |  38.89000015258789 |
|  2 |  38.12999954223633 |
|  3 | 32.859999656677246 |
|  4 |  33.21000003814697 |
|  5 |  38.89000015258789 |
+----+--------------------+
5 rows in set (0.00 sec)
```

Let's combine the two to make sure we get exactly what we expect from the test data. We should first examine the data and try to determine what we should see. In this case, we should see one plant with an average temperature of 38.89 and another with 38.13 (rounding up).

```
mysql> SELECT name, location, AVG(temperature) as avg_temp
    -> FROM plants JOIN readings ON plants.id = readings.id
    -> WHERE DATE(event_time) = CURRENT_DATE() AND plants.climate = 2
    -> GROUP BY plants.id;
+--------------------+----------+-------------------+
| name               | location | avg_temp          |
+--------------------+----------+-------------------+
| Jerusalem Cherry   | deck     | 38.89000015258789 |
| Moses in the Cradle | patio   | 38.12999954223633 |
+--------------------+----------+-------------------+
2 rows in set (0.00 sec)
```

Yes! Now we know that query gets the results we want. What about that other, complex query?

Testing the Complex Query

For the complex query, we want to know the plants that need watering today. You have already seen how to break this query down into parts to construct the SQL statement. Let's see how the database generates the results of each part then combine them to validate the query.

Let's begin with the view we created. Recall, this view returns the ID, soil_status, and count of each soil_status value for those readings taken today. That is, we should see several rows for those plants that have more than one soil status value for today.

```
mysql> SELECT *
    -> FROM soil_status_today;
+----+-------------+------------+
| id | soil_status | num_events |
+----+-------------+------------+
|  1 | DRY         |         10 |
|  2 | OK          |         10 |
|  3 | DRY         |          4 |
|  3 | OK          |          4 |
|  3 | WET         |          2 |
|  4 | OK          |          6 |
|  4 | WET         |          4 |
|  5 | OK          |         10 |
+----+-------------+------------+
8 rows in set (0.00 sec)
```

Great! Now we know the status of each plant's soil moisture for today. Notice some plants have more than one value as their soil moisture changed throughout the day. What we want are those plants that have more "dry" values than other values. But let's slow down a bit.

Recall we used a function to calculate the maximum samples for a given plant taken today. Let's use that function to zero in on the plant IDs in the previous results. In this case, we see several rows that have a higher number of one value for soil_status. Let's use the function to return the percentage of occurrences for each value found. In this case, I will query for all the plants, calculating the percentages so that we can see all the data more easily. What we should see are the same rows as before only with the average samples added.

```
mysql> SELECT id, soil_status, num_events, (num_events/max_samples_today(id)) as
percent_occurrence
    -> FROM soil_status_today;
+----+-------------+------------+--------------------+
| id | soil_status | num_events | percent_occurrence |
+----+-------------+------------+--------------------+
|  1 | DRY         |         10 |             1.0000 |
|  2 | OK          |         10 |             1.0000 |
|  3 | DRY         |          4 |             0.4000 |
|  3 | OK          |          4 |             0.4000 |
|  3 | WET         |          2 |             0.2000 |
|  4 | OK          |          6 |             0.6000 |
|  4 | WET         |          4 |             0.4000 |
|  5 | OK          |         10 |             1.0000 |
+----+-------------+------------+--------------------+
8 rows in set (0.00 sec)
```

Great, seven rows! Now, we're getting somewhere. Now let's limit the output of that result with only those that have a higher than 50 percent occurrence.

```
mysql> SELECT id, soil_status, num_events, (num_events/max_samples_today(id)) as
percent_occurrence
    -> FROM soil_status_today
    -> WHERE (num_events/max_samples_today(id)) > 0.50;
+----+-------------+------------+--------------------+
| id | soil_status | num_events | percent_occurrence |
+----+-------------+------------+--------------------+
|  1 | DRY         |         10 |             1.0000 |
|  2 | OK          |         10 |             1.0000 |
|  4 | OK          |          6 |             0.6000 |
|  5 | OK          |         10 |             1.0000 |
+----+-------------+------------+--------------------+
4 rows in set (0.00 sec)
```

Notice all we did was add a WHERE clause. Now we've got those rows for readings taken today that have a higher than 50 percent occurrence of a single value for soil_status. Let's expand that query one more time and add the condition for soil_status = 'DRY'.

■ **Tip** Did you notice something there? Look at soil_status = 'DRY'. Notice anything weird about that? Yep, it's an enumerated column, and I used one of the values instead of a numeric value like previous examples. As you can see, so long as the values are listed in the enumeration, you can use either the numeric or text value. You will get an error if the text does not match one of the enumerated values.

```
mysql> SELECT id, soil_status, num_events, (num_events/max_samples_today(id)) as
percent_occurrence                           -> FROM soil_status_today
    -> WHERE (num_events/max_samples_today(id)) > 0.50 AND soil_status = 'DRY';
+----+-------------+------------+--------------------+
| id | soil_status | num_events | percent_occurrence |
+----+-------------+------------+--------------------+
|  1 | DRY         |         10 |             1.0000 |
+----+-------------+------------+--------------------+
1 row in set (0.00 sec)
```

Perfect! We're almost there. Now, we want to know the name and location of that plant, id = 1. For this, we add a new join to get the information from the plants table. Notice I use the numeric value for the soil status column.

```
mysql> SELECT plants.id, name, location, soil_status, num_events, (num_events/max_samples_
today(plants.id)) as percent_occurrence
    -> FROM soil_status_today JOIN plants ON soil_status_today.id = plants.id
    -> WHERE (num_events/max_samples_today(plants.id)) > 0.50 AND soil_status = 1;
+----+-----------------+----------+-------------+------------+--------------------+
| id | name            | location | soil_status | num_events | percent_occurrence |
+----+-----------------+----------+-------------+------------+--------------------+
|  1 | Jerusalem Cherry | deck    | DRY         |         10 |             1.0000 |
+----+-----------------+----------+-------------+------------+--------------------+
1 row in set (0.01 sec)
```

Aha! Now, we've got it. Well, almost. We have too much information. But at least we can see the information is correct. Now, we just want the plant name and location. Let's limit that output a bit.

```
mysql> SELECT name, location
    -> FROM soil_status_today JOIN plants ON soil_status_today.id = plants.id
    -> WHERE (num_events/max_samples_today(plants.id)) > 0.50 AND soil_status = 1;
+------------------+----------+
| name             | location |
+------------------+----------+
| Jerusalem Cherry | deck     |
+------------------+----------+
1 row in set (0.00 sec)
```

And there it is! Easy, right? Well, not so much if this was your first attempt, but notice how much easier it is to write the query when we use tools such as views, functions, and aggregate features!

Now that we have seen a complete and working example, let's consider the recommendations and best practices for designing databases for IOT solutions.[15]

Recommendations and Best Practices

Let's review the best practices and recommendations for how you should build you database(s) for your IOT solution. This section presents a number of tips for designing databases. That said, this section cannot cover all there is to learn or know about database design. Indeed, there are many more, especially for enterprise-level database design and management. Thus, it would require several chapters, many pages, and an entire book many times the size of this section to do it justice. However, I feel it is important to close out the discussion of learning to use MySQL and databases in your IOT solutions with a reference for you to review when planning your IOT database design. I list them in no particular order.

- *Use the correct data type*: Do your homework and determine the best data type to use for each column. Avoid the temptation of using all character fields, which can make comparisons return false results or frustrate you as you try to figure out why a view or function fails.

- *Use the smallest data type*: Try to use the smallest data type for your data. For example, avoid using wide character fields. You can always increase the size if you need to (should you start truncating the data). Similarly, use the binary type that matches the maximum value you will store. For example, if you do not need high precision or big numbers, do not use double or long integer data types.

- *Use indexes*: For queries on data that is more than a few dozen rows, consider adding indexes on the frequently queried columns. Indexes can vastly improve joins and complex queries such as ranges if there are indexes on the columns being compared or calculated.

- *Don't store redundant data*: Try to avoid the temptation of storing the same value in multiple tables. It may make your queries easier to write but can make updates difficult. That is, how would you know where every occurrence of the column exists for larger databases? Use master-detail relationships to simplify the data.

---

[15]Actually, adhering to best practices for any database design.

- *Plan your queries*: Always include the questions you want to ask of the database when designing tables. Preparing the queries ahead of time will help with development efforts later.

- *Avoid using SELECT **: While it is easy to just get all the columns for a table, the use of * in the column specification can be suboptimal for large tables or tables with a lot of columns. We saw this earlier in the complex example. The only columns needed were the name and location. Thus, you should resist the temptation to get all the columns and instead specify the columns you want.

- *Use lookup tables for fixed or seldom changed data*: Similar to redundant data, using static (or seldom updated) tables can help reduce the amount of extra data you are passing in result sets and storing. It is much more efficient to store a shorter integer or similar key than to store one or more fields throughout the table.

- *Use the power of the database server whenever possible*: Try to think of ways to offload the processing of aggregates, mathematical functions, and especially computational expensive tasks such as date and string manipulation to the database server. The power of views, functions, triggers, and so on, cannot be understated. It may take some time to become proficient in these concepts, but the payoff can mean using smaller, cheaper components in your IOT network nodes.

- *Use good, coherent names*: Try to use as descriptive and coherent names in your database objects. Resist the temptation to save coding keystrokes by using a, b, c, and so on, as object or column names. If someone other than you tried to figure out what the data describes, they'd be completely lost. It is better to be slightly verbose than terse. Finally, avoid using acronyms or nonstandard abbreviations, which can obfuscate the meaning. For example, what is kdxprt? Kid expert? Nope. I had to ask the designer—it means (in a politically correct manner), "former parent." Don't do that.

- *Use primary keys on master tables*: For any data that can be uniquely identified, use a primary key that can uniquely identify each row. You can use multiple columns in a primary key. For tables where the columns do not uniquely identify the row, you can add a surrogate or artificial primary key using an AUTO_INCREMENT integer data type.

- *Avoid wide tables*: If your table contains 20 or more columns, it is likely it is poorly designed. More specifically in database terms, it is not normalized. Look at the data again and determine whether the columns used can be moved to another table. Look for columns that are reference in nature, do not change frequently, or are redundant.

- *Do not throw away data*: You should always retain the raw data in your tables. If your database stores sensor readings, store the raw values. If you need a calculated column to make reading or queries easier, it is OK to store them like I showed you in the plant-monitoring database, but try to make these columns simple and use a trigger to set the value rather than code on your data or sensor nodes. This isolates the code to a single location and once tested can be relied upon. Having the raw data will allow you to plan queries in the future that you may not have considered requiring the raw data.

- *Avoid storing binary data*: While database systems allow you to store large, binary data (in MySQL a BLOB or binary large object), they are not efficient and a poor choice for tables that store many rows this way, especially if they do not change. For example, if you wanted to store a photo associated with a data item (a row in your master table), you should consider making a field to store a path to the image and store it on the database server. While this creates a new problem—changing the path—it removes the BLOB from your table and can make queries a bit more efficient.

- *Normalize your database*: Normalization is a big deal for a lot of database experts and rightly so, but for the enthusiast and hobbyist getting your database into one of the higher normal forms (from relational database theory) may be far too much work. That said, I encourage you to strive for third normal form,[16] but don't kill yourself getting there. You can ruin a large database quickly trying to over normalize. A small amount of under normalization is permissible if efficiency is not lost to gain simplicity.

- *Design your database before coding*: Always design your database after you've defined the high-level requirements but before you implement the source code. This may sound backward, but it is an excellent habit to form.

- *Test, test, test!* I cannot stress this enough. Time spent testing your database and more importantly your queries with known (test) data will save you a lot of headaches later when you start developing an application to query and present the data.

- *Back up your data*: Once you have your solution running, perform a backup on the data. If you do not have a lot of data, tools such as mysqldump, MySQL Utilities,[17] and similar can make a logical backup of your data in SQL form, which you can restore should you need to do so. If your data is larger, say gigabytes or more, you should consider a commercial backup solution such as MySQL Enterprise Backup.

- *Document your database*: It may seem like extra work if your database contains only a single or small number of tables and little data, but imagine what would happen if your solution ran for years without issue and then one day you needed to add new features or troubleshoot a problem. If you don't know what the database stores or how it produces the results (views, triggers, and so on), you may spend a lot of time digging into false leads. You can document your database in a number of ways. I like to store the SQL statements in a file and write short descriptions of each object. See the example code for this chapter for an example.

Summary

The MySQL database server is a powerful tool. Given its unique placement in the market as the database server for the Internet, it is not surprising that IOT developers (as well as many startup and similar Internet properties) have chosen MySQL for their IOT solutions. Not only is the server robust and easy to use, it is also available as a free community license that you can use to keep your initial investment within budget.

In this chapter, you discovered some of the power of using a database server, how database servers store data, and how to issue commands for creating databases and tables for storing data as well as commands for retrieving that data. While this chapter presents only a small primer on MySQL, you learned how to get started with your own IOT data through the example IOT solution shown. It is likely your own IOT solution will be similar in scope (but perhaps not the same database objects or table layout [schema]).

In the next chapter, you will see how to build a database node using a Raspberry Pi. You will see how the examples in this chapter concerning how and where the data is stored can be leveraged to make a robust MySQL server using a low-cost computer. The process for installing MySQL on other boards is similar.

---

[16]https://en.wikipedia.org/wiki/Database_normalization
[17]http://dev.mysql.com/downloads/utilities/

■ ■ ■ ■

Building Low-Cost MySQL Data Nodes

Data nodes are a key component in IOT solutions. Your solution could use one or more data aggregators to send data to a database server in the cloud or one or more database servers in the solution itself. If your IOT solution uses custom-designed hardware, you may even incorporate a database server on an embedded computing component. Whichever the choice, you need to know more about using a database server in your solution.

At a minimum, you need to know how to get those nodes or components in your solution to send data to the database server. This could be a sensor node with a microcontroller that sends data to a data aggregator, a sensor node that sends data to a data aggregator, or the data aggregator that sends data to the database server.

While there are several choices for a database server including a desktop or server computer, IOT solutions tend to use smaller computing devices like those you saw in Chapter 3 such as a single-board computer. For example, you could use a mini-PC like the pcDuino, a single board computer like the Raspberry Pi, Beaglebone Black, or Intel Galileo.

Since the Raspberry Pi is one of the more popular choices, I will focus on the Raspberry Pi in this chapter, but I include notes about other platforms in case you'd like to use those. Just keep in mind some of these platforms are still evolving and may require more work than the Raspberry Pi. Following the discussion about the Raspberry Pi should give you the background needed for other platforms.

This chapter presents information about how to use data nodes in an IOT solution from a sensor networking point of view, that is, a solution that uses a network of nodes to distribute the processing either for cost or for physical distribution (such as reading sensors around a large agricultural or industrial complex).

You will learn how to create a data node (database server) in this chapter, which features a short introduction to the Raspberry Pi followed by a walk-through of how to set up a MySQL server using a Raspberry Pi. You will also learn how to connect to your database server from your sensor or data aggregation nodes.

Let's begin with a look into the Raspberry Pi.

Introducing the Raspberry Pi

The Raspberry Pi is a small, inexpensive personal computer. Although it lacks the capacity for memory expansion and can't accommodate on-board devices such as CD, DVD, and hard drives,[1] it has everything a simple personal computer requires. There have been several iterations of the Raspberry Pi. The newest version, the Raspberry Pi 2B (`http://raspberrypi.org/products/raspberry-pi-2-model-b/`), has four USB ports, an Ethernet port, HDMI video, and even an audio connector for sound.

---

[1]But can accept USB-based memory sticks and hard drives.

The Raspberry Pi 2B has a micro SD drive[2] that you can use to boot the computer into any of several Linux operating systems. All you need is an HDMI monitor (or DVI with an HDMI-to-DVI adapter), a USB keyboard and mouse, and a 5V power supply, and you're off and running!

■ **Note**   I use the term *micro SD* to refer to a specific media and *SD* to refer to the drive or card in abstract.

You can also power your Raspberry Pi using a USB port on your computer. In this case, you need a USB type A male to micro-USB type B male cable. Plug the type A side into a USB port on your computer and the micro-USB type B side into the Raspberry Pi power port.

There have been many improvements to the Raspberry Pi over the few years it has been around. But the largest improvement is in the area of support. The `Raspberrypi.org` organization has worked very hard to improve the initial experience for new users. There is an easy-to-use and navigate web site that combines all of the old, hard-to-find wikis, lists, charts, and blogs into a central place. You can find just about anything you need to get started on Raspberrypi.org. Figure 6-1 shows an excerpt of the main page.

Figure 6-1. *Raspberrypi.org main page (courtesy of Raspberrypi.org)*

The menu across the top provides links to their extensive blogs with examples and how-to articles, a comprehensive downloads page for all the available operating systems and tools, community Raspberry Pi projects, documentation (help), discussion forums, and resources for teachers, students, and makers.

There is also a link to the Raspberry Pi online shop (see the tag to the right with the Raspberry Pi logo) where you can buy boards, accessories, swag, and more. The shop is based in the United Kingdom, but there are links to online retailers in case you want to find a dealer closer to you.

The Raspberry Pi board is available in several versions and comes as a bare board costing as little as $35 (or $5 for the new Raspberry Pi Zero). It can also be purchased online from electronics vendors such as Sparkfun and Adafruit. Most vendors have a host of accessories that have been tested and verified to work with the Raspberry Pi. These include small monitors, miniature keyboards, and even cases for mounting the board.

---

[2]Micro Secure Digital (micro SD): a small removable memory drive. See `http://en.wikipedia.org/wiki/Secure_Digital`.

In this section, you explore the origins of the Raspberry Pi 2B, take a tour of the hardware connections, and discover what accessories are needed to get starting using the Raspberry Pi.

Noble Origins

The Raspberry Pi was designed to be a platform to explore topics in computer science. The designers saw the need to provide inexpensive, accessible computers that could be programmed to interact with hardware such as servomotors, display devices, and sensors. They also wanted to break the mold of having to spend hundreds of dollars on a personal computer and thus make computers available to a much wider audience.

The designers observed a decline in the experience of students entering computer science curriculums. Instead of having some experience in programming or hardware, students are entering their academic years having little or no experience with working with computer systems, hardware, or programming. Rather, students are well versed in Internet technologies and applications. One of the contributing factors cited is the higher cost and greater sophistication of the personal computer, which means parents are reluctant to let their children experiment on the family PC.

This poses a challenge to academic institutions, which have to adjust their curriculums to make computer science palatable to students. They have had to abandon lower-level hardware and software topics because of students' lack of interest or ability. Students no longer want to study the fundamentals of computer science such as assembly language, operating systems, theory of computation, and concurrent programming. Rather, they want to learn higher-level languages to develop applications and web services. Thus, some academic institutions are no longer offering courses in fundamental computer science.[3] This could lead to a loss of knowledge and skillsets in future generations of computer professionals.

To combat this trend, the designers of the Raspberry Pi felt that, equipped with the right platform, youth could return to experimenting with personal computers as in the days when PCs required a much greater commitment to learning the system and programming it in order to meet your needs. For example, the venerable Commodore 64, Amiga, and early Apple and IBM PC computers had limited software offerings. Having owned a number of these machines, I was exposed to the wonder and discovery of programming at an early age.[4]

WHY IS IT CALLED RASPBERRY PI?

The name was partly derived from design committee contributions and partly chosen to continue a tradition of naming new computing platforms after fruit (think about it). The Pi portion comes from Python, because the designers intended Python to be the language of choice for programming the computer. However, other programming language choices are available.

The Raspberry Pi is an attempt to provide an inexpensive platform that encourages experimentation. The following sections explore more about the Raspberry Pi, including the models available, required accessories, and where to buy the boards.

---

[3]My alma mater has suffered a similar transition. I mourn for the loss of knowledge.
[4]My first real computer was an IBM PCjr. I followed it by building my own IBM PC AT computer, complete with a 10MB hard drive. Ah, those were the glory days of personal computers!

Models
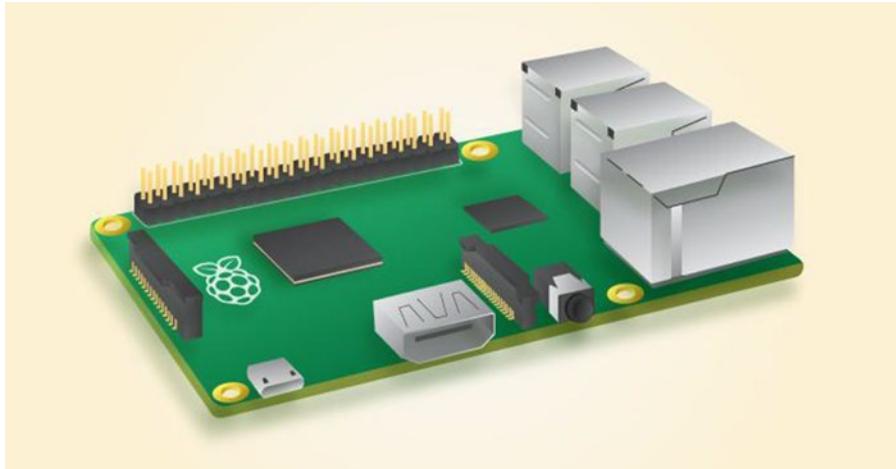
The choices for Raspberry Pi boards have grown to include four models with several versions and iterations. I outline each of the choices here. Figures 6-2 through 6-4 show representations of each.

- *Raspberry Pi 2 Model B*: Second-generation board with a faster processor and 1Gb RAM but retains the layout of the Model B.

- *Raspberry Pi 1 Model B+*: New generation board with more GPIO pins, more USB ports (four), and a micro SD card slot. It also requires a bit less power than older boards.

- *Raspberry Pi Model A+*: Same new features of the Model B but cheaper with fewer USB ports, no Ethernet, and a slightly smaller footprint.

- *Compute Module Development Kit*: Incorporates a new, smaller edge-mounted Raspberry Pi module and expanded GPIO pins (120 instead of 40). It is designed for industrial applications.
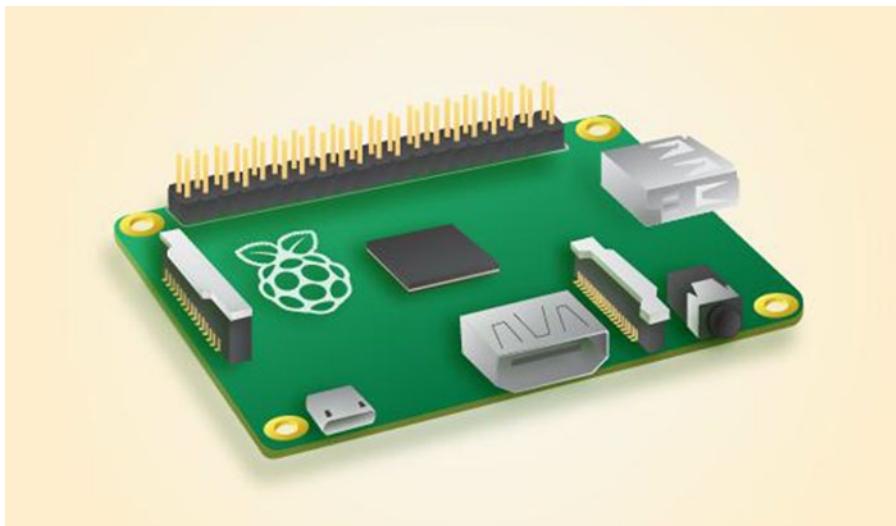
■ **Note**   At the time of this writing, the Raspberry Pi Zero was released. It has the same processor and memory as the original Raspberry Pi but lacks the Ethernet and auxiliary video ports. This was done to keep the cost to an amazingly low $5.00 and a form factor down to the size of a pack of chewing gum.
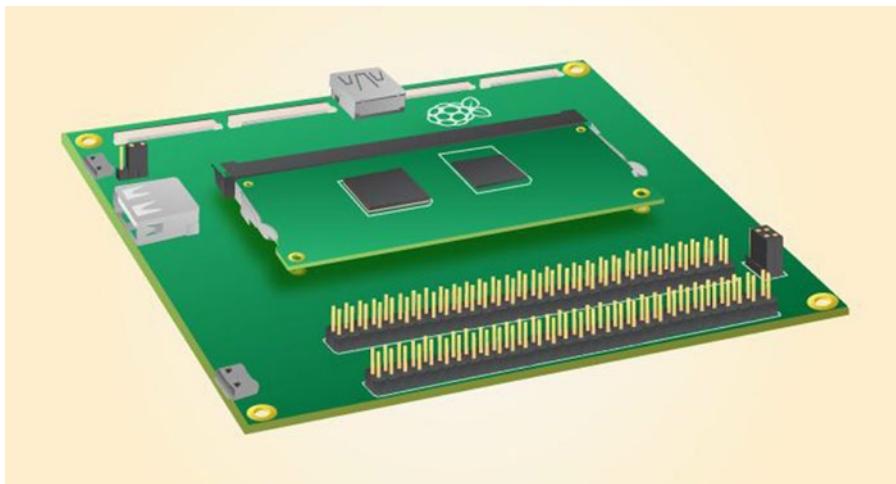
Figure 6-2. *Raspberry Pi 2 Model B and Pi 1 Model B+ (courtesy of Raspberrypi.org)*

Figure 6-3. *Raspberry Pi Model A+ (courtesy of Raspberrypi.org)*

Figure 6-4. *Raspberry Pi Compute Module Development Kit (courtesy of Raspberrypi.org)*

WHAT HAPPENED TO THE $20 RASPBERRY PI?

If you have been following the release of the Raspberry Pi in various media, you have probably heard that the boards were priced at a mere $25. However, most retailers list the Raspberry Pi for $35 or more. Why is that?

The simple answer is the Model A+ is the one priced at $25, whereas the Model B's cost a bit more at $35. This is because the Model B has a few more features specifically Ethernet. If you do not need Ethernet or other B-specific options, you can save a bit by buying the A+.

However, because of supply and demand, you are likely to see average prices for either board (in the United States) at $40 or more. Shop wisely.

Figure 6-2 is a good representation of the Model B series. The Pi 2 and Pi 1 are difficult to distinguish. You must examine the printing on the board itself to determine the differences. There are subtle differences, but they are difficult to see. Fortunately, since they use the same layout, most Model B cases will fit both boards.
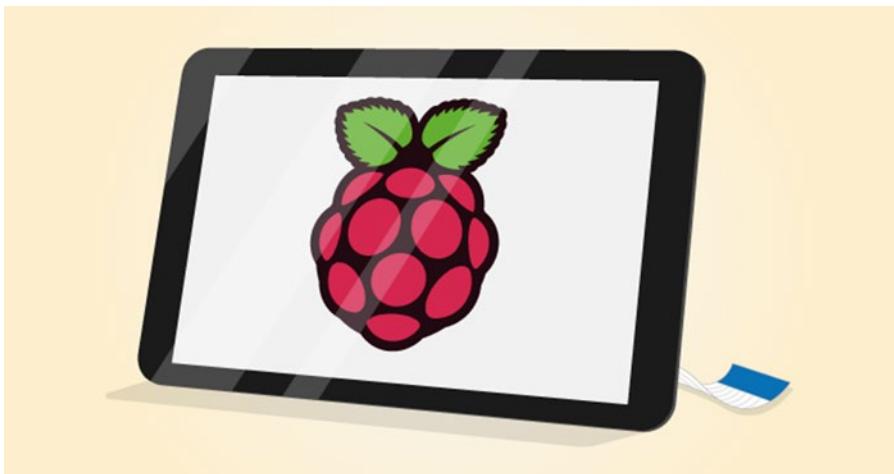
The examples in this chapter and the remaining chapters use the Model B variant.

A Tour of the Board

Not much larger than a deck of playing cards, the Raspberry Pi board contains a number of ports for connecting devices. This section presents a tour of the board. If you want to follow along with your board, hold it with the Raspberry Pi logo face up. I will work around the board clockwise.

In the center of the near side you see an HDMI connector. To the left is a microUSB connector used to supply power to the board, and on the right is an audio port. The power connector is known to be a bit fragile on some boards, so take care plugging and unplugging it. Be sure to avoid putting extra strain on this cable while using your Raspberry Pi.

The HDMI port is the primary way to connect a monitor. However, there is a small ribbon cable connector on the left called a DSI video connector. The 7" Raspberry Pi Touch Display (http://element14.com/community/docs/DOC-78156?ICID=hp-7inchpidisplay-ban) can be connected here to provide a really nice, small tablet-like experience. Figure 6-5 shows the Raspberry Pi Touch Display.

Figure 6-5. *Raspberry Pi 7" Touch Display (courtesy of Raspberrypi.org)*

There is also a camera connector located behind the Ethernet port for connecting a camera (useful for many applications).

■ **Note**   Some enterprising makers have developed enclosures for the new monitor including a tablet form factor that can be 3D printed. See `http://thingiverse.com` for the latest prototypes and designs.

On the left side bottom of the board is the micro SD card slot on the bottom.

On the far side and on top of the board is the general-purpose input/output (GPIO) header (a double row pins), which can be used to attach to sensors and other devices.

On the right side of the board is where most of the connectors are placed. There are four USB connectors and the Ethernet connector. An external-powered USB hub connected to the USB ports on the Raspberry Pi can power some boards, but it is recommended that you use a dedicated power supply connected to the micro-USB connector.

Take a moment to examine the top and bottom faces of the board. As you can see, components are mounted on both sides. This is a departure from most boards that have components on only one side. The primary reason the Raspberry Pi has components on both sides is that it uses multiple layers for trace runs. This permits the board to be much smaller and enables the use of both surfaces. This is probably the most compelling reason to consider using a case—to protect the components on the bottom of the board and thus avoid shorts and board failure.

Required Accessories

The Raspberry Pi is sold as a bare system board with no case, power supply, or peripherals. Depending on how you plan to use the Raspberry Pi, you need a few commonly available accessories. If you have been accumulating spares like me, a quick rummage through your stores may locate most of what you need.

If you want to use the Raspberry Pi in console mode (no graphical user interface), you need a USB power supply, a keyboard, and an HDMI monitor (or the 7" Touch Display). The power supply should have a minimal rating of 700mA or greater operating at 5V. If you want to use the Raspberry Pi with a graphical user interface, you also need a pointing device (such as a mouse).

If you have to purchase these items, stick to the commonly available brands and models without extra features. For example, avoid the latest multifunction keyboard and mouse. Chances are, they require drivers that are not available for the various operating system choices for the Raspberry Pi.

You also must have a micro SD card. I recommend a 8GB or higher version. Recall that the micro SD is the only onboard storage medium available. You will need to put the operating system on the card, and any files you create will be stored on the card. I will demonstrate this in a later section.

If you want to use sound in your applications, you also need a set of powered speakers that accept a standard 3.5mm audio jack. Finally, if you want to connect your Raspberry Pi to the Internet, you need an Ethernet cable or a Raspberry Pi–compatible USB Wi-Fi dongle.

HOW CAN I TELL IF MY DEVICE WILL WORK?

If you want to make sure your device will work with the Raspberry Pi, the simplest thing to do is try it! If you prefer not to take chances, you can check the Raspberry Pi hardware compatibility list at http://elinux.org/RPi_VerifiedPeripherals. This list contains many devices and commentary from various users in the community who have tested the devices. If you are just starting out with the Raspberry Pi, look for devices that require little or no extra configuration or drivers.
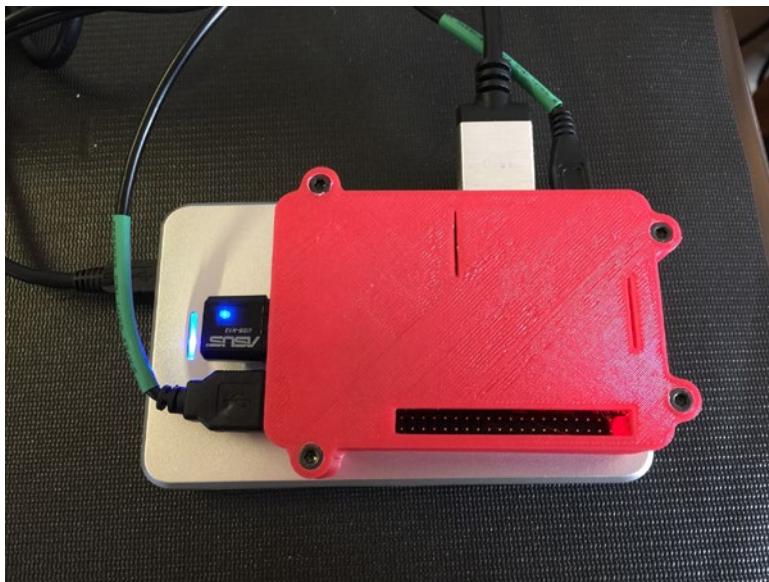
Recommended Accessories

I highly recommend at least adding small rubber or silicone self-adhesive bumpers to keep the board off your desk. On the bottom of the board are many sharp prongs that can come into contact with conductive materials, which can lead to shorts or, worse, a blown Raspberry Pi. These bumpers are available at most home-improvement and hardware stores.

If you plan to move the board from room to room or you want to ensure that your Raspberry Pi is well protected against accidental damage, you should consider purchasing a case to house the board. Many cases are available, ranging from simple snap-together models to models made from laser-cut acrylic or even milled aluminum. The following list includes several excellent choices, ranging from inexpensive to top-of-the-line luxury cases:

- *Raspberry Pi 2B+ Clear case*: http://sparkfun.com/products/12996

- *Pi Tin*: http://sparkfun.com/products/13102

- *Bel-Aire*: http://makershed.com/products/the-bel-aire

- *Adafruit Pi Box*: http://adafruit.com/products/1985

- *Ninja Pibow*: http://adafruit.com/products/2081

- *Unibody Aluminum Case*: http://adafruit.com/products/2198

Another option is to print your own case with a 3D printer. If you do not have a 3D printer, you may find one in your local library or community college or perhaps a friend of a friend. If you ask nicely, chances are most 3D printer enthusiasts would be happy to print you a case. Indeed, there are many such case designs available for downloading and printing. Figure 6-6 shows one I printed on my 3D printer.

Figure 6-6. *3D-printed Raspberry Pi 2 Model B+ case*

■ **Tip** If you plan to experiment with the GPIO pins or require access to the power test pins or the other ports located on the interior of the board, you may want to consider either using the self-adhesive bumper option or ordering a case that has an open top to make access easier. Some cases are prone to breakage if opened and closed frequently.

Aside from a case, you should also consider purchasing (or pulling from your spares) a powered USB hub. The USB hub power module should be 700–1000mA. A powered hub is required if you plan to use USB devices that draw a lot of power, such as a USB hard drive or a USB soft missile launcher.

■ **Caution** Because the board is small, it is tempting to use it in precarious places like in a moving vehicle or on a messy desk. Ensure that your Raspberry Pi is in a secure location. The power, HDMI, and micro SD card slots seem to the most vulnerable connectors.

Where to Buy

The Raspberry Pi has been available in Europe for some time. It is getting easier to find, but few brick-and-mortar stores stock the Raspberry Pi. Fortunately, a number of online retailers stock it, as well as a host of accessories that are known to work with the Raspberry Pi. The following are some of the more popular online retailers with links to their Raspberry Pi catalog entry:

- *Sparkfun*: http://sparkfun.com/categories/233

- *Adafruit*: http://adafruit.com/category/105

- *Maker Shed*: http://makershed.com/collections/raspberry-pi
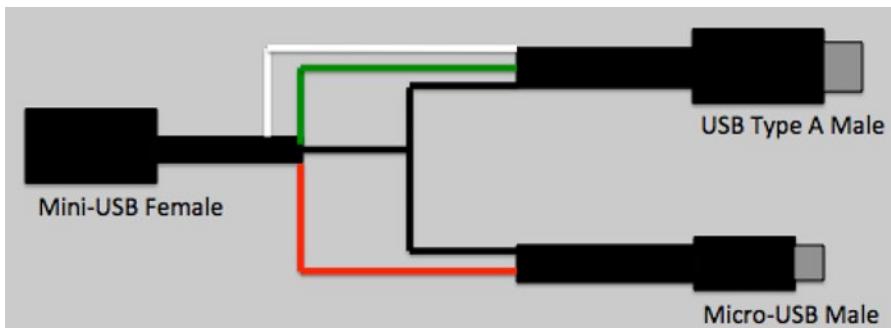
A RASPBERRY PI LAPTOP?

The Raspberry Pi has made a significant contribution to physical computing. Not only does it enable more sophisticated sensor nodes, but it also makes a decent lightweight general-purpose computer. With a proper monitor, mouse, and storage device, you can do most Internet and modest productivity tasks. In fact, some people have replaced their home desktop computer with a Raspberry Pi!

If you are like me and you need to be able to work from anywhere,[5] using a Raspberry Pi may not be very convenient, given that you must have a separate monitor and keyboard. Wouldn't it be great if you could take your Raspberry Pi with you? Well, now you can!

What you need is a surplus Atrix Lapdock from Motorola. Originally designed to allow the Aria phone to be used as a laptop, the Lapdock provides an 11.6" HDMI monitor, a USB keyboard, a mouse, a two-port USB hub, and speakers. More important, it is battery powered and can easily power the Raspberry Pi. The Lapdock has mini-HDMI and mini-USB ports that can be connected to the Raspberry Pi without modifying the Lapdock.

But there is a catch: you must purchase a mini-HDMI female-to-female adapter and a mini-HDMI male to HDMI male cable[6] and build your own Frankenstein USB cable from a micro-USB extension cable and a type A USB cable. The custom cable is needed to allow the Raspberry Pi to use the USB keyboard and mouse as well as power the board. The following figure shows how the cable is constructed. You can find a detailed tutorial video at www.adafruit.com/blog/2012/09/10/cables-adapters-for-the-atrix-raspberry-pi-laptop/.
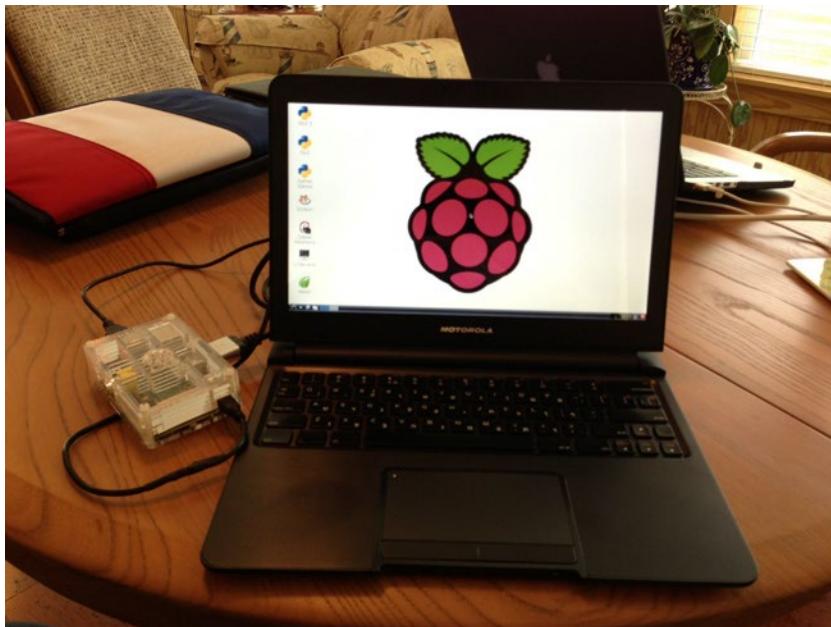
To build the cable, cut a normal USB type A connector from a standard USB cable,[7] bisect a standard micro-USB extension cable, and splice the wires as shown here. You can abandon the wires in the ends that have no connections shown. The micro-USB male connector will be used to power the Raspberry Pi, and the USB type A male connector will provide connectivity to the Lapdock keyboard and mouse. Once you have this cable made, you're ready to go. For the best results, I recommend a sturdy case to mount your Raspberry Pi so as not to damage it during transport.

---

[5]Places like your couch, favorite recliner, patio, coffee bar, and so on.

[6]Be sure to select a cable that supports device sensing. If your Lapdock does not power on when the Raspberry Pi is connected, it is most likely the HDMI cable. Try another cable.

[7]Be sure to get permission before "borrowing" a cable from a friend or spouse—it won't be usable as a standard cable when you're finished modifying it.

To use the Raspberry Pi laptop, start by connecting the HDMI cable, then any peripherals (like a hard drive), and then the USB cable; open the lid of the Lapdock. Within a few seconds, your new laptop is ready to go! The next figure shows the ports on the rear of the Lapdock under the folding door, and the one after that shows the laptop in action.

Not only can you use this solution for your Raspberry Pi, but also several other boards work equally well. I have successfully used the Lapdock to power other boards. Indeed, the Lapdock makes an excellent pcDuino laptop! Even if your board doesn't support the keyboard or mouse, the HDMI screen is handy.

The best part of this project is the cost. You can find used and surplus Lapdock on auction sites and similar electronics clearance stores. For example, on eBay the Lapdock is priced at about $60. You can also find the cables online or at most electronics stores. However, the female-to-female mini-HDMI adapter is a bit harder to find. I was able to purchase one on eBay from a dealer in China. Shipping was surprisingly fast, and the cost was reasonable. My cost for a mobile Raspberry Pi (not including the Raspberry Pi) was less than $100.

The next section presents a short tutorial on getting started using the Raspberry Pi. If you have already learned how to use the Raspberry Pi, you can skim the section to see the latest improvements in getting your Raspberry Pi up and running.

Raspberry Pi Tutorial

The following sections present a short tutorial on getting started with your new Raspberry Pi, from a bare board to a fully operational platform. A number of excellent works cover this topic in much greater detail. If you find yourself stuck or wanting to know more about beginning to use the Raspberry Pi and more about the Raspbian operating system, see *Learn Raspberry Pi with Linux* by Peter Membrey and David Hows (Apress, 2012). If you want to know more about using the Raspberry Pi in hardware projects, an excellent resource is *Practical Raspberry Pi* by Brendan Horan (Apress, 2013).

As mentioned in the "Required Accessories" section, you need a micro SD card, a USB power supply rated at 700mA or better with a male micro-USB connector, a keyboard, a mouse (optional), and an HDMI monitor, an HDMI TV, or a DVI monitor with an HDMI adapter. However, before you can boot your Raspberry Pi and bask in its brilliance, you need to create a boot image for your micro SD card.

Choosing a Boot Image (Operating System)

The first thing you need to do is decide which operating system variant you want to use. There are several excellent choices, including the standard Raspbian "Jessie" variant. Each is available as a compressed file called an *image* or *card image*. You can find a list of recommended images along with links to download each on the Raspberry Pi foundation download page: www.raspberrypi.org/downloads. The following images are available at the site:

- *Raspbian (Jessie)*: Debian-based official operating system and contains a graphical user interface (Lightweight X11 Desktop Environment [LXDE]), development tools, and rudimentary multimedia features.

- *Ubuntu Mate*: Features the Ubuntu desktop and a scaled-down version of the Ubuntu operating system. If you are familiar with Ubuntu, you will feel at home with this version.

- *Snappy Ubuntu Core*: Developer's edition of core Ubuntu system; same as Mate but with addition of the developer core utilities.

- *Windows 10 IOT Core*: Windows 10 for the IOT. Microsoft's premier IOT operating system. Yes, it does look and feel like Windows, but without the heavy graphical user interface.

- *OSMC*: Open source media center. Build yourself a media center.

- *OpenElec*: Open embedded Linux entertainment center. Another media center option.

- *PiNet*: Classroom management system. A special edition for educators using the Raspberry Pi in the curriculum.

- *RISC OS*: Non-Linux, Unix-like operating system. If you know what IBM AIX is or you've used other Unix operating systems, you'll recognize this beastie.

■ **Tip**   If you are just starting with the Raspberry Pi, you should use the Raspbian image. This image is also recommended for the examples in this book.

There are a few other image choices, including a special variant of the Raspbian image from Adafruit. Adafruit calls its image *occidentals* and includes a number of applications and utilities preinstalled, including Wi-Fi support and several utilities. Some Raspberry Pi examples—especially those from Adafruit— require the occidentals image. You can find out more about the image and download it at `http://learn.adafruit.com/adafruit-raspberry-pi-educational-linux-distro/overview`.

There are two methods for installing the boot image. First, you can use the automated, graphical user interface platform named New Out Of the Box Software (NOOBS[8]), or you can install your image from scratch onto a micro SD drive. Both require downloading and formatting the micro SD drive.

If you are just starting out, the NOOBS solution is by far the easiest. It will take a bit longer to get going (but not much) and simplifies the process. Aside from formatting the micro SD card, everything is automated. I present both options in the following sections.

Using NOOBS

NOOBS is by far the best way to get your Raspberry Pi up and running. With NOOBS, you download a base installer image that contains Raspbian Jessie. You can choose to install it or configure NOOBS to download one of the other images and install it. But first, you have to get the NOOBS boot image and copy it to your micro SD drive.

Begin by downloading the NOOBS installer from `http://raspberrypi.org/downloads/noobs/`. You will see two options, a network installer (sometimes referred to as the *offline installer*) that includes the Raspbian image or a base image that does not contain any operating systems. This base image is what you would use if you wanted to use the automated installer with an operating system not already included such as Adafruit's version.

■ **Tip**   If your download bandwidth is limited, online retailers offer a preconfigured micro SD card that includes NOOB. In fact, you can often find micro SD cards with any of the popular Raspberry Pi operating systems. Just plug it in and go. They usually cost a few dollars more than a blank card of the same size.[9]

---

[8]Not to be confused with noob, which is a bit derogatory. See `https://en.wiktionary.org/wiki/noob`.
[9]The operating system is free. You're just paying for the convenience of someone having formatted and installed the image on the card for you.

Once you've downloaded the installer (to date about 1.4Gb), you will need to format a micro SD card of at least 8Gb. You can use a variety of ways to do this depending on your desktop platform. If you use Mac OS X, you can format the drive with Disk Utility. Or you can use the SD Formatter 4.0 utility available for Windows or Mac OS X (http://sdcard.org/downloads/formatter_4/). Simply download the application and install it. Then insert your micro SD card in your card reader and launch the application. Once you verify you've selected the correct media, enter a name for the card (I used NOOBS) and click format. Figure 6-7 shows the SDFormatter application.
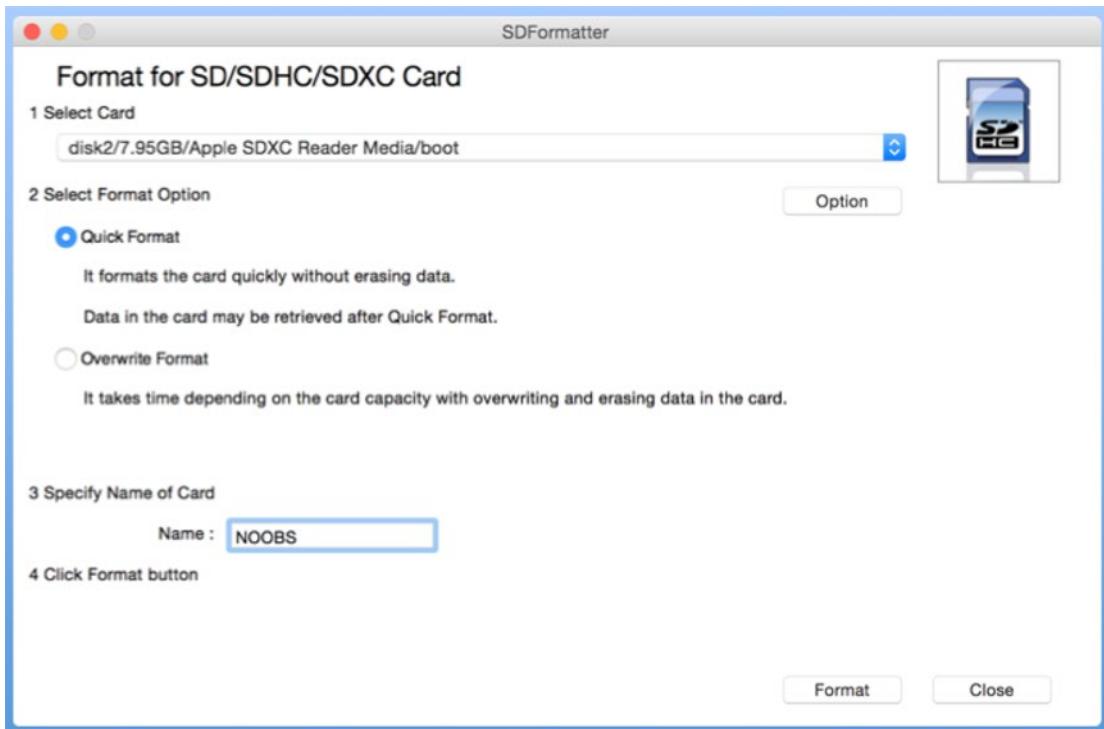
Figure 6-7. *SDFormatter 4.0*
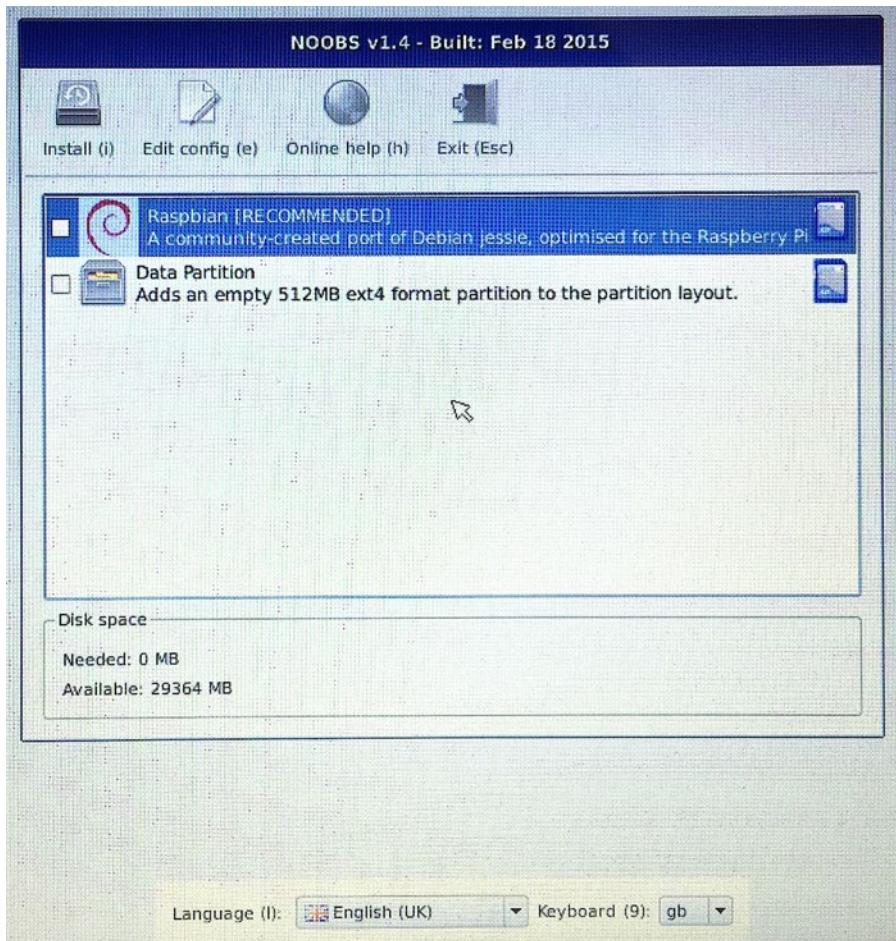
BUT, I DON'T HAVE AN SD CARD READER!

You must have an SD card reader/writer connected to your computer. Some systems have SD card drives built in (Lenovo laptops, Apple laptops and desktops, and so on). If you do not have an SD card reader, you can find USB SD card readers anywhere electronics or photo equipment is sold. Most readers can accept various formats including micro SD or micro SD via a micro SD to SD adapter.

Once you've formatted the micro SD card, you now must copy the contents of the NOOBS image to the card. Right-click the file you downloaded and choose the option to unzip or unarchive the file. This will create a folder containing the NOOBS image. Copy all of those files (not the outside folder) to the SD card and eject it. You are now ready to boot into NOOBS and install your operating system. When this process has finished, safely remove the SD card and insert it into your Raspberry Pi.

You are now ready to hook up all of your peripherals. I like to keep things simple and connect only a monitor, keyboard, and (for NOOBS) a mouse. If you want to download an operating system other than Raspbian, you will also need to connect your Raspberry Pi to your network.

Once your Raspberry Pi powers on, you will see a scrolling display of various messages. This is normal and may scroll for some time before NOOBS starts. When NOOBS is loaded, you will see a screen similar to Figure 6-8.
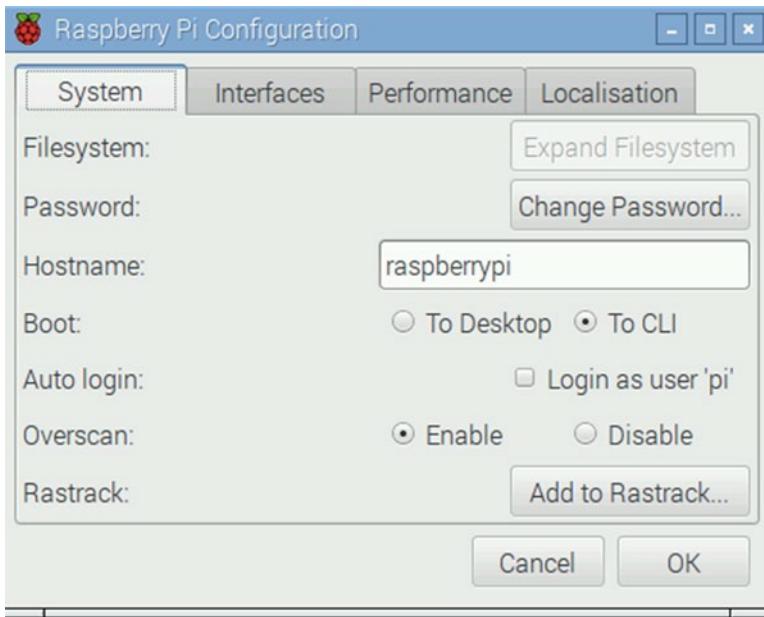
Figure 6-8. *NOOBS startup screen*

Notice you will see the Raspbian image in the list of operating systems. To install it, just tick the checkbox beside the thumbnail then click Install. However, note the two boxes at the bottom. This sets the language and keyboard for use in NOOBS. It does not affect the setup of Raspbian.

Once you initiate the install, you will see a series of dialogs as Raspbian begins its installation. This could take a while. The good news is the dialogs provide a lot of useful information to help you get started. You will learn about how to log in to Raspbian, tips for configuring and customizing, and suggestions for how to get the most out of your experience.

When installation finishes, click OK on the completed dialog and then wait for the Raspberry Pi to reboot into Raspbian. On the first boot, you may see the Raspberry Pi Configuration dialog. The configuration dialog is used to set the time and date for your region, enable hardware like a camera board, create users, change the password, and more. Figure 6-9 shows the configuration dialog.

Figure 6-9. *Raspbian configuration dialog*

■ **Tip** You can also run the console-based configuration utility by opening a terminal and running the command `raspi-config`.

You will see four tabs that you can use to change settings for the system. I explain each briefly in the following list along with recommended settings for each. Once you have made your changes, click OK to close the dialog. Depending on which settings you choose, you may be asked to reboot.

- *System*: Board controls for the system. Use this panel to change the root password (highly recommended), hostname (optional), type of boot (use command-line interface [CLI] if you want to set up the Raspberry Pi to boot headless, and automatic login (not recommended).

- *Interfaces*: Used to enable system and hardware services such as the camera, SSH (recommended), and hardware interfaces for the GPIO header.

- *Performance*: Used to make changes to how the processor performs. You can choose to overclock (run the CPU faster), but I do not recommend this setting for a Raspberry Pi that will host a database or web server (or both).

- *Localisation*: Used to set the default language, keyboard, and date and time. If you change nothing else, be sure to set these to your local settings.

■ **Tip** The default login for Raspbian Jessie uses the username `pi` and password `raspberry`. I recommend changing this in the Raspberry Pi Configuration dialog.

To shut down or reboot Raspbian, click the menu, and then choose Shutdown. You will see a prompt for rebooting, shutting down, or returning to the command line. If you are at the command line, use the command `shutdown -h now` to shut down the system.

Installing Boot Image on a Micro SD Card

The process of installing a boot image involves choosing an image, downloading it, and then copying it to your micro SD card. The following sections detail the steps involved. This is a manual process that is a bit more complicated than the NOOBS option but not overly so.

Once you select an image and download it, you first unzip the file and then copy it to your SD card. There are a variety of ways to do this. The following sections describe some simplified methods for a variety of platforms.

Windows

To create the SD card image on Windows, you can use the Win32 Disk Imager software from Launchpad (`https://launchpad.net/win32-image-writer`). Download this file, and install it on your system. Unzip the image if you haven't already, and then insert your SD card into your SD card reader/writer. Launch the Win32 Disk Imager application, select the image in the top box, and then click WRITE to copy the image to the SD.

■ **Caution** The copy process overwrites anything already on the SD card, so be sure to copy those photos to your hard drive first!

Mac OS X

To create the SD card image on the Mac, download the image and unzip it. Insert your SD card into your SD card reader/writer. Be sure the card is formatted with FAT32. Next, open the System report (hint: use the Apple menu and then select About this Mac).

Click the card reader if you have a built-in card reader, or navigate through the USB menu and find the SD card. Take note of the disk number. For example, it could be disk4.

Next, open Disk Utility and unmount the SD card. You need to do this to allow Disk Utility to mount and connect to the card. Now things get a bit messy. Open a terminal, and run the following command, substituting the disk number for *n* and the path and name of the image file for <image_file>:

```
sudo dd if=<image_file> of=/dev/diskn bs=1m
```

At this point, you should see the disk-drive indicator flash (if there is one), and you need to be patient. This step can run for some time with no user feedback. You will know it is complete when the command prompt is displayed again.

Linux

To create the SD card image using Linux, you need to know the device name for the SD card reader. Execute the following command to see the devices currently mounted:

```
df -h
```

Next, insert the SD card or connect a card reader, and wait for the system to recognize it. Run the command again:

```
df -h
```

Take a moment to examine the list and compare it to the first execution. The "extra" device is your SD card reader. Take note of the device name (for example, /dev/sdc1). The number is the partition number. So, /dev/sdc1 is partition 1, and the device is /dev/sdc. Next, unmount the device (I will use the previous example).

```
umount /dev/sdc1
```

Use the following command to write the image, substituting the device name for <device> and path and name of the image file for <image_file> (for example, /dev/sdc and my_image.img):

```
sudo dd bs=4M if=<image_file> of=<device>.
```

At this point, you should see the disk-drive indicator flash (if there is one), and you may need to be patient. This step can run for some time with no user feedback. You will know it is complete when the command prompt is displayed again.

Booting Up

To boot your Raspberry Pi, insert the SD card with the new image and plug in your peripherals. Wait to plug in the USB power last. Because the Raspberry Pi has no On/Off switch, it will start as soon as power is supplied. The system bootstraps and then starts loading the OS. You see a long list of statements that communicate the status of each subsystem as it is loaded. You don't have to try to read or even understand all the rows presented,[10] but you should pay attention to any errors or warnings. When the boot sequence is complete, you see a command prompt, as shown in Figure 6-10.

---

[10]They go by so fast; it is unlikely you can read them anyway. Basically, they're noise unless there is an error, and those usually appear in the last few lines displayed.
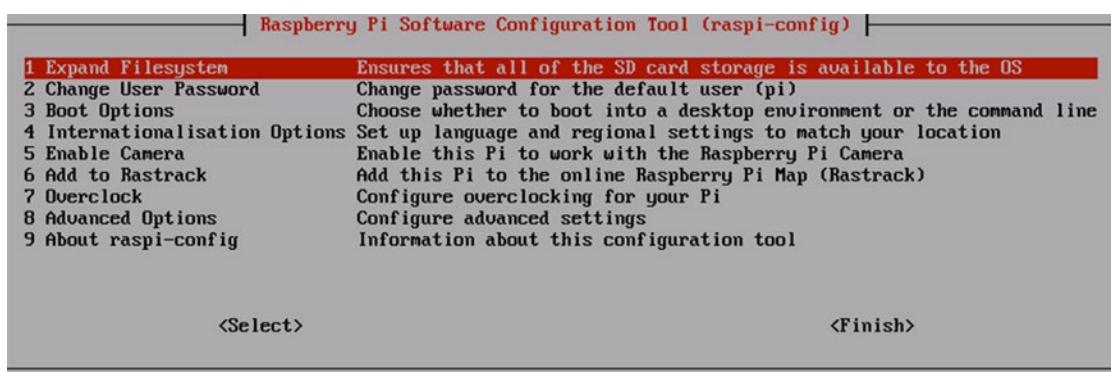
```
OK  ] Started Avahi mDNS/DNS-SD Stack.
       Starting System Logging Service...
       Starting Permit User Sessions...
OK  ] Started Restore Sound Card State.
OK  ] Started /etc/rc.local Compatibility.
OK  ] Started LSB: Brings up/down network automatically.
OK  ] Started LSB: Autogenerate and use a swap file.
OK  ] Started LSB: triggerhappy hotkey daemon.
OK  ] Started LSB: Start NTP daemon.
OK  ] Started Permit User Sessions.
OK  ] Started System Logging Service.
       Starting Wait for Plymouth Boot Screen to Quit...
       Starting Terminate Plymouth Boot Screen...
OK  ] Started Wait for Plymouth Boot Screen to Quit.
OK  ] Started Terminate Plymouth Boot Screen.
OK  ] Started Login Service.
       Starting Getty on tty1...
OK  ] Started Getty on tty1.
       Starting Serial Getty on ttyAMA0...
OK  ] Started Serial Getty on ttyAMA0.
OK  ] Reached target Login Prompts.
OK  ] Started LSB: IPv4 DHCP client with IPv4LL support.
OK  ] Started LSB: Start and stop the mysql database server daemon.
OK  ] Reached target Multi-User System.
       Starting Update UTMP about System Runlevel Changes...
```

Figure 6-10. *Example boot sequence*[11]

You may be prompted to enter a username and password. The default user is simply *pi*, and the password is *raspberry* (no quotes, all lowercase). Enter that at the prompts, and the Raspberry Pi presents you with the configuration menu shown in Figure 6-11. If you do not see this, you can launch it by typing the command raspi-config.


```
┤ Raspberry Pi Software Configuration Tool (raspi-config) ├
1 Expand Filesystem          Ensures that all of the SD card storage is available to the OS
2 Change User Password       Change password for the default user (pi)
3 Boot Options               Choose whether to boot into a desktop environment or the command line
4 Internationalisation Options Set up language and regional settings to match your location
5 Enable Camera              Enable this Pi to work with the Raspberry Pi Camera
6 Add to Rastrack            Add this Pi to the online Raspberry Pi Map (Rastrack)
7 Overclock                  Configure overclocking for your Pi
8 Advanced Options           Configure advanced settings
9 About raspi-config         Information about this configuration tool


             <Select>                                        <Finish>
```

Figure 6-11. *Raspberry Pi configuration menu*

---

[11]Raspberry Pi images were generated with fbgrab. You can install it with sudo apt-get install fbgrab.

The configuration menu displays a list of common initial options you may want to set when using the Raspberry Pi. It is loaded on the first boot for convenience. You can navigate among the options using the up and down arrow keys and select the action buttons using the Tab key. The menu items are described briefly here:

- *Expand Filesystem*: Use the full space on the SD card.

- *Change User Password*: Change the password for the Pi user. Use this if your Raspberry Pi will be connected to a network and especially if it is accessible from the Internet.

- *Boot Options*: Enable/disable boot to the GUI windowing system.

- *Overscan*: Change how the video signal is sent to the monitor/TV. Use this if your output image from the Raspberry Pi does not fill the available display area.

- *Internationalisation Options*: Change keyboard mapping (country/language specific), time zone, and language, which sets country/language-specific display modes for how time, currency, dates, and so on, are displayed.

- *Enable Camera*: Turn on/enable the camera module.

- *Add to Rastrack*: Add your Raspberry Pi to the Pi Map, a global indicator of the Raspberry Pi's in the world.

- *Overclock*: Change the CPU timing settings (also called *speed*) for the system. Experts only. This is not needed for normal Raspberry Pi use.

- *Advanced Options*: Enable/disable various system services such as SSH.

- *About* `raspi-config`: Get information about how to use this tool.

■ **Note** Future releases of the configuration menu will include additional options. Once you have connected your Raspberry Pi to the Internet and executed the Update option, it is a good idea to check the menu for new options.

The first time you boot your system, you should use a few of these options. At a minimum, you should set the root file system to use the entire SD card space, change the keyboard setup, set your locale and time zone, and, if you want to be able to remotely log in, enable the SSH server.

When you first initialize an image on an SD card, the process does not use the entire space available. The Expand Filesystem option does this for you. In some cases, the system will be rebooted when the operation is complete, so make sure you don't have other things running before executing this option.

Setting the keyboard, locale, and time zone enables you to use the Raspberry Pi in a manner you are used to with a PC. In particular, your keyboard will have the special symbols where you expect them; dates, time, and similar values will be displayed correctly; and your clock will be set the correct local time. These operations may not require a reboot. You should set these prior to using the Raspberry Pi in earnest.

On future boots, the system will start, and, once you are logged in, it will be in terminal mode (unless you selected the option to start in the windowing environment). From here, you can explore the system using command-line utilities or start the graphical user interface with `startx`. Take some time and explore the system before proceeding. If you want to restart the configuration session, use the command `sudo raspi-config`.

Once your Raspberry Pi is running and you have spent time exploring and learning the basics for system administration, you are ready to start experimenting with hardware.

SD CARD CORRUPTION

Imagine this scenario. You're working away on creating files, downloading documents, and so on. Your productivity is high, and you're enjoying your new low-cost, super-cool Raspberry Pi. Now imagine the power cable accidentally gets kicked out of the wall, and your Raspberry Pi loses power. No big deal, yes? Well, most of the time.

The SD card is not as robust as your hard drive. You may already know that it is unwise to power off a Linux system abruptly, because doing so can cause file corruption. Well, on the Raspberry Pi it can cause a complete loss of your disk image. Symptoms range from minor read errors to an inability to boot or load the image on bootstrap. This can happen—and there have been reports from others that it has happened more than once.

That is not to say all SD cards are bad or that the Raspberry Pi has issues. The corruption on accidental power-off is a side effect of the type of media. Some have reported that certain SD cards are more prone to this than others. The best thing you can do to protect yourself is to use an SD card that is known to work with Raspberry Pi and be sure to power the system down with the `sudo shutdown -h now` command—and never, ever power off the system in any other manner.

You can also make a backup of your SD card. See http://elinux.org/RPi_Beginners#Backup_your_ SD_card for more details.

Now that you know how to get your Raspberry Pi set up and running, let's now discover how to turn it into a database server for your IOT solution.

MySQL Installation and Setup

It is time to get your hands dirty and work some magic on your unsuspecting Raspberry Pi! Let's begin by adding a USB hard drive to it. Depending on the size of your data, you may want to seriously consider doing this.

That is, if your data will be small (never more than a few megabytes), you may be fine using MySQL from your boot image SD card. However, if you want to ensure that you do not run out of space and keep your data separate from your boot image (always a good idea), you should mount a USB drive that automatically connects on boot. This section explains how to do this in detail.

■ **Tip** Be sure you use a good-quality powered USB hub to host your external drive. This is especially important if you are using a traditional spindle drive because it consumes a lot more power. Connecting your external drive directly to the Raspberry Pi may rob it of power and cause untold frustration. Symptoms include random reboot (always a pleasant surprise), failed commands, data loss, and so on. Always be sure you have plenty of power for your peripherals as well as your Raspberry Pi.

The choice of what disk to use is up to you. You can use a USB flash drive, which should work fine if it has plenty of space and is of sufficient speed (most newer models are fast enough). You can also use a solid-state drive (SSD) if you have an extra one or want to keep power usage and heat to a minimum. On the other hand, you may have an extra hard drive lying around that can be pressed into service. This section's example uses a surplus 250GB laptop hard drive mounted in a typical USB hard drive enclosure.

■ **Tip**   Using an external hard drive—either an SSD or traditional spindle drive—is much faster than accessing data on a flash drive. It is also typically cheaper per unit (gigabyte) or, as I mentioned, can be easily obtained from surplus.

Partitioning and Formatting the Drive

Before you can use a new or an existing drive with a file system incompatible with the Raspberry Pi, you must partition and format the drive. I find it easier to do this on my desktop computer and suggest you do the same. Thus, the following assumes the external drive has a single FAT (or FAT32) partition. That isn't so important because we will delete it and create a new partition with the ext4 file system for optimal performance.

Begin by connecting the drive to the Raspberry Pi. Then determine what drives are attached by using the fdisk -l command to see the available disks connected. You should see your hard drive listed as /dev/sda if you have a standard Raspbian image. If you use a different image or your device is labeled differently, use the address from your system in the following steps.

Once you identify the disk, launch fdisk again with the device as an option, as shown in Listing 6-1.

Listing 6-1. Partitioning a Hard Disk on Raspberry Pi

```
pi@raspberrypi ~ $ sudo fdisk /dev/sda

Welcome to fdisk (util-linux 2.25.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): m

Help:

  Generic
   d   delete a partition
   l   list known partition types
   n   add a new partition
   p   print the partition table
   t   change a partition type
   v   verify the partition table

  Misc
   m   print this menu
   x   extra functionality (experts only)

  Save & Exit
   w   write table to disk and exit
   q   quit without saving changes
```

```
  Create a new label
   g   create a new empty GPT partition table
   G   create a new empty SGI (IRIX) partition table
   o   create a new empty DOS partition table
   s   create a new empty Sun partition table

Command (m for help): p
Disk /dev/sda: 111.8 GiB, 120034123776 bytes, 234441648 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 790E7C68-F089-45C7-A9E9-D7C2CA56BB31

Command (m for help): n
Partition number (1-128, default 1): 1
First sector (34-234441614, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-234441614, default 234441614):

Created a new partition 1 of type 'Linux filesystem' and of size 111.8 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

There are a number of things going on here. I've highlighted the steps in bold. Notice first I show the help menu for the utility with the m command per the prompt. Next, I use the p command to print the partition table verifying that there is no partition there. If you had partitions defined and wanted to delete them, you'd use the d command to do so.

■ **Caution**   If you have a partition on your drive that has data you want to keep, abort now and copy the data to another drive first. The following steps erase all data on the drive!

You then create a new partition using the command n and accept the defaults to use all the free space. To check your work, you can use the p command to print the device partition table and metadata. It shows (and confirms) the new partition.

If you are worried that you may have made a mistake, do not panic! The great thing about fdisk is that it doesn't write or change the disk until you tell it to with the w or write command. In the example, you issue the w command to write the partition table. To see a full list of the commands available, you can use the h command or run man fdisk.

■ **Tip**   For all Linux commands, you can view the manual file by using the command man <application>.

The next step is to format the drive with the ext4 file system. This is easy and requires only one command: `mkfs` (make file system). You pass it the device name. If you recall, this is `/dev/sda1`. Even though you created a new partition, it is still the first partition because there is only one on the drive. If you are attempting to use a different partition, be sure to use the correct number! The command may take a few minutes to run, depending on the size of your drive. The following example shows the command in action:

```
pi@raspberrypi ~ $ sudo mkfs.ext4 /dev/sda1
mke2fs 1.42.12 (29-Aug-2014)
Creating filesystem with 29304945 4k blocks and 7331840 inodes
Filesystem UUID: 0285ba01-3880-4ee5-8a19-7f47404f1500
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000, 7962624, 11239424, 20480000, 23887872

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

Now you have a new partition, and it has been properly formatted. The next step is associating the drive with a mount point on the boot image and then connecting that drive on boot so you don't have to do anything to use the drive each time you start your Raspberry Pi.

Setting Up Automatic Drive Mounting

External drives in Linux are connected (mounted) with `mount` and disconnected (unmounted) with `umount`. Unlike with some operating systems, it is generally a bad idea to unplug your USB drive without unmounting it first. Likewise, you must mount the drive before you can use it. This section shows the steps needed to mount the drive and to make the drive mount automatically on each boot.

I begin with a discussion of the preliminary steps to get the drive mounted and ready for automatic mounting. These include creating a folder under the `/media` folder to mount the drive (called a *mount point*), changing permissions to the folder to allow access, and executing some optional steps to tune the drive.

```
pi@raspberrypi ~ $ sudo mkdir -p /media/HDD
pi@raspberrypi ~ $ sudo chmod 755 /media/HDD
pi@raspberrypi ~ $ sudo tune2fs -m 0 /dev/sda1
tune2fs 1.42.12 (29-Aug-2014)
Setting reserved blocks percentage to 0% (0 blocks)
pi@raspberrypi ~ $ sudo tune2fs -L MYSQL /dev/sda1
tune2fs 1.42.12 (29-Aug-2014)
pi@raspberrypi ~ $ sudo mount /dev/sda1 /media/HDD
```

These commands are easy to discern and are basic file and folder commands. However, the tuning steps using `tune2fs` (tune file system) are used to first reset the number of blocks used for privileged access (which saves a bit of space) and then label the drive as MYSQL. Again, these are optional, and you may skip them if you like.

■ **Tip** You can unmount the drive with `sudo umount /dev/sda1`.

At this point, the drive is accessible and ready to be used. You can change to the /media/HDD folder and create files or do whatever you'd like. Now let's complete the task of setting up the drive for automatic mounting.

The best way to do this is to refer to the drive by its universally unique identifier (UUID). This is assigned to this drive and only this drive. You can tell the operating system to mount the drive with a specific UUID to a specific mount point (/media/HDD).

Remember the /dev/sda device name from earlier? If you plugged your drive in to another hub port—or, better still, if there are other drives connected to your device and you unmount and then mount them—the device name may not be the same the next time you boot! The UUID helps you determine which drive is your data drive, frees you from having to keep the drive plugged in to a specific port, and allows you to use other drives without fear of breaking your MySQL installation if the drive is given a different device name.

To get the UUID, use the blkid (block ID) application.

```
pi@raspberrypi ~ $ sudo blkid
/dev/mmcblk0: PTUUID="000575b3" PTTYPE="dos"
/dev/mmcblk0p1: LABEL="RECOVERY" UUID="0761-F2EA" TYPE="vfat" PARTUUID="000575b3-01"
/dev/mmcblk0p3: LABEL="SETTINGS" UUID="13062706-1a48-47bc-9f3a-0ded961267e4" TYPE="ext4"
PARTUUID="000575b3-03"
/dev/mmcblk0p5: SEC_TYPE="msdos" LABEL="boot" UUID="07D7-3A9D" TYPE="vfat"
PARTUUID="000575b3-05"
/dev/mmcblk0p6: LABEL="root" UUID="c9d8e201-90e5-4d6b-9c8f-92d658fec13c" TYPE="ext4"
PARTUUID="000575b3-06"
/dev/sda1: LABEL="MYSQL" UUID="0285ba01-3880-4ee5-8a19-7f47404f1500" TYPE="ext4"
PARTUUID="ab7357a8-536a-4015-a36d-f80280c2efd1"
```

Notice the line in bold. Wow! That's a big string. A UUID is a 128-byte (character) string. Copy it for the next step.

To set up automatic drive mapping, you use a feature called *static information* about the file system (fstab). This consists of a file located in the /etc folder on your system. You can edit the file however you like. If you are from the old school of Linux or Unix, you may choose to use vi.[12] The resulting file is as follows:

```
pi@raspberrypi ~ $ sudo nano /etc/fstab
proc            /proc           proc    defaults          0       0
/dev/mmcblk0p5  /boot           vfat    defaults          0       2
/dev/mmcblk0p6  /               ext4    defaults,noatime  0       1
UUID=0285ba01-3880-4ee5-8a19-7f47404f1500 /media/HDD ext4 defaults.noatime 0 0
# a swapfile is not a swap partition, no line here
#   use  dphys-swapfile swap[on|off]  for that
```

The line you add is shown in bold. Here you simply add the UUID, mount point, file system, and options. That's it! You can reboot your Raspberry Pi using the following command and watch the screen as the messages scroll. Eventually, you see that the drive is mounted. If there is ever an error, you can see it in the bootup sequence.

```
$ sudo shutdown -r now
```

---

[12]What does *vi* mean? If you've ever had the pleasure of trying to learn it for the first time, you may think it means "virtually impossible," because the commands are terse (by design) and difficult to remember. But seriously, *vi* is short for *vim* or *Vi Improved* text editor. The name suggests that the original editor may very well have been *completely* impossible to use!

Now you are ready to build a MySQL database server! The following section details the steps needed to do this using your Raspberry Pi.

Installing MySQL Server

Turning a Raspberry Pi into a MySQL database server is easy. This section shows you how to install MySQL and then how to move its default data directory from your boot image to the new external drive you connected in the previous section.

The steps involved include updating your aptitude base (the package manager) and then installing MySQL. Although the process is rather lengthy, I felt it best to show you the entire thing in case your base image is different or you encounter errors.

Installing MySQL

To install MySQL or any software not already in your base image, you must be connected to the Internet. If you have not already done so, connect your Raspberry Pi to the Internet using the Ethernet port or a wireless networking device.

As you may recall, you are using the Raspbian Jessie distribution, which is Debian-based. If you use some other distribution, it may have a different package manager, and the commands in this section may not work. In that case, you should be able to find similar commands for your distribution.

Let's begin with updating the package manager package headers. This is always a good idea, especially if you are using a distribution that was released more than a few months ago. The command `apt-get update` tells the system to download the latest headers from known host distributions. This ensures that you get the latest version of whatever software you are installing.

After that, installing the software is as simple as telling aptitude to install it. The trick is knowing the correct name. In this case, you're looking for `mysql-server`. Listing 6-5 shows the steps for updating aptitude and installing MySQL. (I have omitted some lines for brevity.) In addition to entering the commands, you are asked to reply to the prompt asking if it is OK to download MySQL and its prerequisites and to enter a password for the root user for MySQL.

■ **Note** When you see the password *secret* in the examples, it is used as a placeholder for whatever password you have chosen—it is not explicitly the word *secret*.

Let's begin by updating the package manager with the sudo `apt-get update` command, as shown here:

```
pi@raspberrypi ~ $ sudo apt-get update
Get:1 http://archive.raspberrypi.org jessie InRelease [13.2 kB]
Get:2 http://mirrordirector.raspbian.org jessie InRelease [15.0 kB]
Get:3 http://archive.raspberrypi.org jessie/main Sources [22.4 kB]
Get:4 http://mirrordirector.raspbian.org jessie/main armhf Packages [8,961 kB]
Get:5 http://archive.raspberrypi.org jessie/ui Sources [5,197 B]
Get:6 http://archive.raspberrypi.org jessie/main armhf Packages [60.2 kB]
Get:7 http://archive.raspberrypi.org jessie/ui armhf Packages [7,639 B]
Get:8 http://mirrordirector.raspbian.org jessie/contrib armhf Packages [37.4 kB]
Get:9 http://mirrordirector.raspbian.org jessie/non-free armhf Packages [70.2 kB]
...
Fetched 9,194 kB in 1min 11s (129 kB/s)
Reading package lists... Done
```

Next, let's install MySQL with the `sudo apt-get install mysql-server` command as shown next. Once you begin the MySQL installation, you may be prompted for setting the root user password for MySQL. Be sure to choose a password you will remember.

```
pi@raspberrypi ~ $ sudo apt-get install mysql-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

The following extra packages will be installed:

```
libaio1 libdbd-mysql-perl libdbi-perl libhtml-template-perl libmysqlclient18 libterm-
readkey-perl mysql-client-5.5 mysql-common
mysql-server-5.5 mysql-server-core-5.5
```

Suggested packages:

```
libclone-perl libmldbm-perl libnet-daemon-perl libsql-statement-perl libipc-sharedcache-perl
mailx tinyca
```

The following NEW packages will be installed:

```
libaio1 libdbd-mysql-perl libdbi-perl libhtml-template-perl libmysqlclient18 libterm-
readkey-perl mysql-client-5.5 mysql-common
mysql-server mysql-server-5.5 mysql-server-core-5.5
0 upgraded, 11 newly installed, 0 to remove and 3 not upgraded.
Need to get 8,121 kB of archives.
After this operation, 88.8 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://mirrordirector.raspbian.org/raspbian/ jessie/main libaio1 armhf 0.3.110-1 [9,228 B]
Get:2 http://mirrordirector.raspbian.org/raspbian/ jessie/main mysql-common all
      5.5.44-0+deb8u1 [74.3 kB]
Get:3 http://mirrordirector.raspbian.org/raspbian/ jessie/main libmysqlclient18 armhf
      5.5.44-0+deb8u1 [616 kB]
...
Setting up libaio1:armhf (0.3.110-1) ...
Setting up libmysqlclient18:armhf (5.5.44-0+deb8u1) ...
Setting up libdbi-perl (1.631-3+b1) ...
Setting up libdbd-mysql-perl (4.028-2+b1) ...
Setting up libterm-readkey-perl (2.32-1+b2) ...
Setting up mysql-client-5.5 (5.5.44-0+deb8u1) ...
Setting up mysql-server-core-5.5 (5.5.44-0+deb8u1) ...
Setting up mysql-server-5.5 (5.5.44-0+deb8u1) ...
151001 12:58:57 [Warning] Using unique option prefix key_buffer instead of key_buffer_size
is deprecated and will be removed in a future release. Please use the full name instead.
151001 12:58:57 [Note] /usr/sbin/mysqld (mysqld 5.5.44-0+deb8u1) starting as process 18455 ...
Setting up libhtml-template-perl (2.95-1) ...
Setting up mysql-server (5.5.44-0+deb8u1) ...
Processing triggers for libc-bin (2.19-18+deb8u1) ...
Processing triggers for systemd (215-17+deb8u2) ...
```

WHAT IF IT DOESN'T WORK?

Although highly unlikely, if it all goes completely wonky,[13] you can remove the MySQL installation bundle with the following commands. They uninstall every package and remove any files created by the installation.

```
sudo apt-get autoremove mysql-server mysql-server-5.5
sudo apt-get purge mysql-server mysql-server-5.5
```

Once you've done this, you can try the install steps again and correct your mistake.

Now that MySQL is installed, let's use the MySQL console and try to connect to the server. The command is mysql -uroot -p<password>, where <password> is the password you supplied when you installed MySQL. Listing 6-2 shows a successful connection to the new MySQL server. I executed some commands to test things and to gather information for the next step. Notice that the MySQL console displays the version of the MySQL server as well as a short name for the platform. In this case, I was connected to a MySQL 5.5.28-1 server on a Debian platform.

Listing 6-2. Connecting to MySQL

```
pi@raspberrypi ~ $ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 5.5.44-0+deb8u1 (Raspbian)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.00 sec)
```

---

[13]A highly technical term for when computers don't do what you think they should.

```
mysql> SHOW VARIABLES LIKE '%dir%';
+-----------------------------------------+---------------------------+
| Variable_name                           | Value                     |
+-----------------------------------------+---------------------------+
| basedir                                 | /usr                      |
| binlog_direct_non_transactional_updates | OFF                       |
| character_sets_dir                      | /usr/share/mysql/charsets/ |
| datadir                                 | /var/lib/mysql/           |
| innodb_data_home_dir                    |                           |
| innodb_log_group_home_dir               | ./                        |
| innodb_max_dirty_pages_pct              | 75                        |
| lc_messages_dir                         | /usr/share/mysql/         |
| plugin_dir                              | /usr/lib/mysql/plugin/    |
| slave_load_tmpdir                       | /tmp                      |
| tmpdir                                  | /tmp                      |
+-----------------------------------------+---------------------------+
11 rows in set (0.00 sec)

mysql>
```

In the example, I issued the SHOW DATABASES command to see the list of databases and the SHOW VARIABLES command to show all variables containing the name dir. Notice the datadir output from the last command: this is the location of your data.

In the next section, you tell MySQL to use the external drive instead for storing your databases and data.

Moving the Data Directory to the External Drive

Recall that you want to use MySQL to store your sensor data. As such, the sensor data may grow in volume and over time may consume a lot of space. Rather than risk filling up your boot image SD, which is normally only a few gigabytes, you can use an external drive to save the data. This section shows you how to tell MySQL to change its default location for saving data.

The steps involved require stopping the MySQL server, changing its configuration, and then restarting the server. Finally, you test the change to ensure that all new data is being saved in the new location. Begin by stopping the MySQL server.

```
$ sudo /etc/init.d/mysql stop
```

You must create a folder for the new data directory.

```
$ sudo mkdir /media/HDD/mysql
```

Now you copy the existing data directory and its contents to the new folder. Notice that you copy only the data and not the entire MySQL installation, which is unnecessary.

```
$ sudo cp -R /var/lib/mysql/*  /media/HDD/mysql
$ chown -R mysql mysql /media/HDD/mysql/
```

Next you edit the configuration file for MySQL. In this case, you change the datadir line to read datadir = /media/HDD/mysql. It is also a good idea to comment out the bind-address line to permit access to MySQL from other systems on the network.

```
$ sudo vi /etc/mysql/my.cnf
```

There is one last step. You must change the owner and group to the MySQL user who was created on installation. Here is the correct command:

```
$ sudo chown -R mysql:mysql /media/HDD/mysql
```

Now you restart MySQL.

```
$ sudo /etc/init.d/mysql start
```

You can determine whether the changes worked by connecting to MySQL, creating a new database, and then checking to see whether the new folder was created on the external drive, as shown in Listing 6-3.

Listing 6-3. Testing the New Data Directory

```
pi@raspberrypi ~ $ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 5.5.44-0+deb8u1 (Raspbian)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE TESTME;
Query OK, 1 row affected (0.00 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
| testme             |
+--------------------+
5 rows in set (0.00 sec)

mysql> quit
```

You can check to see that the database is created by displaying the file structure using the following command:

```
$ sudo ls -lsa /media/HDD/mysql
```

What you should see in the `mysql` folder is a separate folder for each database. Indeed, you should see the folder for the new database created represented as the folder `testme`. Well, there you have it—a new MySQL database server running on a Raspberry Pi!

WHAT ABOUT OVERHEATING?

Concerns about overheating a Raspberry Pi are mainly for those who attempt overclocking and other risky modifications; you should worry if your Raspberry Pi is run continuously. Typically you run a database server 24/7, shutting it down only for maintenance.

If you are concerned about overheating, you can add heat sinks to your Raspberry Pi's major components for a reasonable cost (about $15). However, I have not seen any issues with running a Raspberry Pi indefinitely if it is housed in an enclosure that permits heat dissipation and it is placed in a climate-controlled environment. A definitive answer to this question has been provided by one of the founders himself (see `www.youtube.com/watch?v=Sz8NMp4MgG0`).

Now that we have our Raspberry Pi configured and MySQL installed and working, let's now see how we can connect our devices to the database server for saving our data. But first, let me discuss some alternatives for the Raspberry Pi.

Other Platforms

While I focused on the Raspberry Pi for demonstrating how to build a low-cost MySQL database server, the Raspberry Pi isn't the only choice. Indeed, you can use any number of low-cost computers and embedded platforms for hosting a MySQL server. In this section, I present three alternatives: the BeagleBone Black, pcDuino, and Intel Galileo.

While the process is similar on all of these, there are some small differences and other things to consider. Thus, I present each in the following sections in a condensed overview. Having read the Raspberry Pi tutorial previously, you should be able to accomplish the following with ease.

BeagleBone Black

Recall from Chapter 3, the BeagleBone Black is a lower-cost version of the original BeagleBone. It comes with an onboard bootable Linux operating system preconfigured. Like the Raspberry Pi, it hosts a number of ports including USB ports for connecting memory devices.

Be sure to connect your BeagleBone Black to your network before powering on. I used the onboard Ethernet port and found it more than adequate for accessing the board remotely. The only issue you may have is discovering which IP address your board is using. I recommend using a port scanner. There are many such applications available for most platforms. Use that IP address and remote into your BeagleBone Black with `ssh root@<IP address>`.

■ **Note** The default root password on the BeagleBone Black is blank.

Installing MySQL

The MySQL installation on BeagleBone Black is similar to the Raspberry Pi example earlier except the commands are a little different but we execute them in the same order. The following shows the commands you execute. We must first update the local packages and package headers. Use the following command to do this:

```
$ opkg update
```

Next, we install MySQL with the following command:

```
$ opkg install mysql5
```

This process will take a while beginning with downloading a number of packages and supporting libraries. Once the installation is complete, issue the following command to launch MySQL:

```
$ /etc/init.d/mysqld start
```

Depending on the date of your BeagleBone Black's preinstalled operating system, you may see an error similar to the following:

```
/etc/init.d/mysqld: line 3: /etc/default/rcS: No such file or directory
BeagleBone - mysqld error.png
```

The problem is an error in the script that starts MySQL. We edit the file with the following command and comment out the third line in the file (more specifically, the line that reads /etc/default/rcS). Just put a # in the first column to comment it out.

```
$ vi /etc/init.d/mysqld
```

Save the file and then restart MySQL as follows:

```
$ /etc/init.d/mysqld start
```

■ **Note**   There is no sudo on the BeagleBone Black default installation. To shut down, simply use the shutdown –h now command.

Configuring the Hard Drive

Configuring the hard drive on the BeagleBone Black uses the same commands as those on the Raspberry Pi. That is, you use fdisk to create a partition and mkfs to create the file system.

pcDuino

The pcDuino is a unique board. The newest versions support the A20 or later processors (multicore and a bit faster than other boards). The board also has many connectors including an onboard SATA port for supporting a SATA hard drive. Perhaps the most interesting and indeed the reason for the name are the board supports Arduino-compatible shields. As you saw in Chapter 3, this allows us to develop our Arduino solutions on a single device.

However, the really nice part is the pcDuino's onboard bootable operating system is a version of Ubuntu 12. This means it will behave similarly to a full installation of Ubuntu. Indeed, as a regular Ubuntu user (second choice to Mac OS X), I found myself right at home on the pcDuino, especially when I use my Lapdock connected to the pcDuino.

That is, with the pcDuino connected to a monitor, keyboard, and mouse, it operates very much the same way as a laptop. It's nearly as fast too! Plus, the onboard Wi-Fi capabilities of the pcDuino 3B make using the board convenient (no Ethernet cable strung across the room).

Since the pcDuino runs Ubuntu, installing MySQL on a pcDuino is the same process as installing MySQL on most any other Ubuntu machine. You can even use the same documentation and examples found elsewhere on the Internet. For pcDuino-specific documentation, see http://linksprite.com/?page_id=874.

■ **Tip** The default user for the pcDuino is ubuntu with password ubuntu.

Installing MySQL

The MySQL installation on the pcDuino requires the following commands. We begin with updating the packages and package headers as follows:

```
$ sudo apt-get update
```

Next, we install MySQL with the following command:

```
$ sudo apt-get install mysql-server
```

You will have to select a root user password twice during the install, but that's it! All you have left to do is configure MySQL to your needs. For example, you may want to edit the my.cnf file and set up your databases as described in the previous chapter.

Configuring the Hard Drive

Configuring the hard drive on the pcDuino is, once again, similar to the other platforms. Since the platform is a bit different, I will walk you through the process.

Begin by plugging the USB drive into the USB port. When the drive is recognized (the access LED may blink a few times and then go solid), execute fdisk much like we did on the Raspberry Pi to create a partition, mkfs to format, and then mount the drive. Finally, set up the drive using the same commands as we used on the Raspberry Pi to make the mount persistent and transfer the MySQL database datadir to the drive as shown previously.

Intel Galileo

The Intel Galileo, sister board to the Intel Einstein platform, is another board that includes headers that accept Arduino shields. Like the pcDuino and the newer Arduino boards that run a Linux operating system such as the Yun, the Intel Galileo provides access to the Arduino either from Linux or via a serial connection to your desktop computer. Thus, like the pcDuino, you can use it to write Arduino sketches as well as an embedded node in your IOT solution.

The Intel Galileo's onboard bootable Linux is minimal, but I recommend downloading the latest Intel SD-Card Linux Image (currently located at http://intel.com/support/galileo/sb/CS-035101.htm), uncompressing and installing it on a bootable micro SD drive (you can find instructions at https://software.intel.com/en-us/creating-bootable-micro-sd-card-for-intel-galileo-board), and then booting the Galileo from the micro SD card.

You can connect your Galileo directly to your network via an Ethernet cable and, like the BeagleBone Black example, use a port scanner to locate the IP address. Unfortunately, there isn't an HDMI display port on the Galileo, so using a monitor and keyboard isn't an option.

Also, the Galileo's Linux image is a bit on the lean side, and the packages built are not as complete as other platforms. In fact, we will have to use an alternative package repository to install MySQL.

Installing MySQL

The MySQL installation on the Intel Galileo is nearly the same as the BeagleBone Black, but as you shall see, it isn't as clean. One difference is there is no base MySQL package available for the default SD-Card Linux image. Fortunately, AlexT[14] has done the work for us. All you need to do is modify the package locations and perform the update. The following shows all the steps for completeness. I recommend starting with a clean boot of the SD-Card Linux image from Intel.

■ **Note**   There is no root password for the SD-Card Linux image on the Galileo.

We begin by updating the package location file named SSS with the following command and data. Just paste this into the empty file.

```
$ vi /etc/opkg/base-feeds.conf
src/gz all      http://repo.opkg.net/galileo/repo/all
src/gz clanton http://repo.opkg.net/galileo/repo/clanton
src/gz i586     http://repo.opkg.net/galileo/repo/i586
```

We then update the local packages and package headers. Use the following command to do this:

```
$ opkg update
```

Next, we install MySQL with the following command:

```
$ opkg install mysql5
```

This process will take a while beginning with downloading a number of packages and supporting libraries. If you get an error, you may need to overwrite the uclibc files with the following command and then restart the installation:

```
$ opkg install --force-overwrite uclibc
```

---

[14]http://alextgalileo.altervista.org/package-repo-configuration-instructions.html

Once the installation is complete, you should be able to start the MySQL server as follows. If you see errors such as the following, you may need to do some extra work. I saw this on at least one Galileo board after an aborted installation. You may not need these steps, but I include them in case you get stuck by this issue. What happened was the installation failed before it created the special user mysql, which is common for most Linux installations. But again, it is easy to fix.

```
$ /etc/init.d/mysqld start
$ 010101 00:27:46 mysqld_safe Logging to '/var/log/mysqld.err'.
chown: unknown user mysql
010101 00:27:46 mysqld_safe Starting mysqld daemon with databases from /var/mysql
010101 00:27:47 mysqld_safe mysqld from pid file /var/lib/mysql/mysqld.pid ended
```

To fix this, simply create the mysql user as follows and start the server again. If you'd like to read more about this process, see the preconfiguration steps in the source code installation section in the online MySQL reference manual (http://dev.mysql.com/doc/refman/5.6/en/installing-source-distribution.html).

```
$ groupadd mysql
$ useradd -r -g mysql mysql
$ chown -R mysql /var/lib/mysql
$ chgrp -R mysql /var/lib/mysql
$ mysql_install_db --user=mysql
$ /etc/init.d/mysqld start
```

Configuring the Hard Drive

Configuring the hard drive on the Galileo is, once again, similar to the other platforms. Since the platform is a bit different, I will walk you through the process.

Begin by plugging the USB drive into the USB port. When the drive is recognized (the access LED may blink a few times and then go solid), execute fdisk much like we did on the Raspberry Pi to create a partition, mkfs to format, and then mount the drive. Finally, set up the drive using the same commands as we used on the Raspberry Pi to make the mount persistent and transfer the MySQL database datadir to the drive as shown previously.

DOES IT MATTER WHAT VERSION OF MYSQL I USE?

You may be wondering about the version of MySQL you should use. While the latest version of MySQL is 5.7, it is unlikely you will find this version in the package list or repositories for the Linux and similar operating systems for boards like those described here. Fortunately, most packages and repositories have either MySQL 5.1 or 5.5 available, which should be sufficient for most IOT solutions, and you are unlikely to need the latest features of MySQL.

However, if you do need the latest features for compatibility or conformity, you may need to download the source code directly from Oracle and build the installation locally. Or you could search for a precompile binary or even an installation package for your platform. While they're rare, I have encountered these packages from time to time. Don't be afraid to ask people on a support forum for your board for help. Chances are someone out there has already created what you need.

MySQL Clients: How to Connect and Save Data

You have already seen how to connect to the MySQL server with the MySQL client. That tool is an interactive tool where we can execute queries, but it isn't helpful for saving data from our sensors or data nodes. What we need is something called a *connector*. A connector is a programming module designed to permit our sketches (from an Arduino) or scripts or programs to send data to the database server. Connectors also allow us to query the database server to get data from the server.

I will cover two primary connectors you are likely to encounter when developing your own IOT solutions. I present each as a tutorial that you can use to follow along with your own hardware. I begin with a connector for use with the Arduino (Connector/Arduino) and then present a connector for use in writing Python scripts (Connector/Python).

DATABASE CONNECTORS FOR MYSQL

There are many database connectors for MySQL. Oracle supplies a number of database connectors for a variety of languages. The following are the current database connectors available for download from http://dev.mysql.com/downloads/connector/:

- *Connector/ODBC*: Standard ODBC compliant
- *Connector/Net*: Windows .Net platforms
- *Connector/J*: Java applications
- *Connector/Python*: Python applications
- *Connector/C++*: Standardized C++ applications
- *Connector/C* (`libmysql`): C applications
- *MySQL native driver for PHP* (`mysqlnd`): PHP 5.3 or newer connector
- *Connector/Arduino*: Arduino sketches

As you can see, there is a connector for just about any programming language you are likely to encounter—and now there is even one for the Arduino!

Introducing Connector/Arduino

With a new database connector made specifically for the Arduino, you can connect your Arduino project directly to a MySQL server without using an intermediate computer or a web-based service. Having direct access to a database server means you can store data acquired from your project in a database. You can also check values stored in tables on the server. The connector allows you to keep your IOT solution local to your facility—it can even be disconnected from the Internet or any other external network.

Saving your data in a database not only preserves the data for analysis at a later time but also means your project can feed data to more complex applications. Better still, if you have projects that use large data volumes for calculations or lookups, you can store the data on the server and retrieve only the data you need for the calculation or operation—all without taking up large blocks of memory on your Arduino. Clearly, this opens a whole new avenue of Arduino projects!

The database connector is named Connector/Arduino. It implements the MySQL client communication protocol (called a *database connector*) in a library built for the Arduino platform. Henceforth I refer to Connector/Arduino when discussing general concepts and features and refer to the actual source code as the Connector/Arduino library, the connector, or simply the library.

Sketches (programs) written to use the library permit you to encode SQL statements to insert data and run small queries to return data from the database (for example, using a lookup table).

You may be wondering how a microcontroller with limited memory and processing power can possibly support the code to insert data into a MySQL server. You can do this because the protocol for communicating with a MySQL server is not only well known and documented but also specifically designed to be lightweight. This is one of the small details that make MySQL attractive to embedded developers.

To communicate with MySQL, the Arduino must be connected to the MySQL server via a network. To do so, the Arduino must use an Ethernet or Wi-Fi shield and be connected to a network or subnet that can connect to the database server (you can even connect across the Internet). The library is compatible with most new Arduino Ethernet, Wi-Fi, and compatible clone shields that support the standard Ethernet library.

■ **Note** Compatibility isn't a hardware requirement so much as a software library limitation. That is, if you use a networking device that uses either the included Ethernet library or a library based on the standard `Ethernet.Client` class, your hardware should be compatible. Some of the newer, low-cost Ethernet modules may not be compatible.

There is a lot you can do with Connector/Arduino. What follows is a short primer on getting started with the connector. If you need more help or want a more in-depth look into the library as well as more examples, download the reference manual from https://github.com/ChuckBell/MySQL_Connector_Arduino/blob/master/extras/MySQL_Connector_Arduino_Reference_Manual.pdf.

WHAT ABOUT MEMORY?

Connector/Arduino is implemented as an Arduino library. Although the protocol is lightweight, the library does consume some memory. In fact, the library requires about 20KB of flash memory to load. Thus, it requires the ATmega328 or similar processor with 32KB of flash memory.

That may seem like there isn't a lot of space for programming your solution, but as it turns out, you really don't need that much for most sensors. If you do, you can always step up to a newer Arduino with more memory. For example, the latest Arduino, the Due, has 512KB of memory for program code. Based on that, a mere 20KB is an insignificant amount of overhead.

The library is open source, licensed as GPLv2, and owned by Oracle Corporation. Thus, any modifications to the library that you intend to share must meet the GPLv2 license. Although it is not an officially supported product of Oracle or MySQL, you can use the library under the GPLv2.
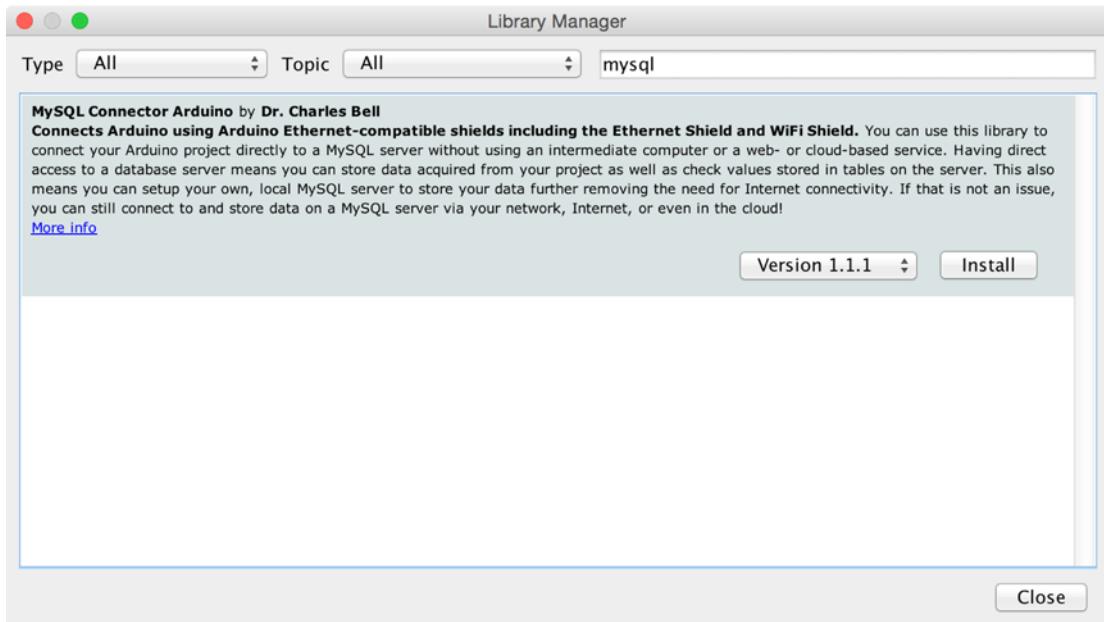
■ **Tip** There is a MySQL forum for discussing the connector. See http://forums.mysql.com/list.php?175. If you get stuck and need some help, check the forum for possible answers.

Installing Connector/Arduino

There are two ways to get and install Connector/Arduino. The first and the recommended method is to use the Library Manager to search for and install the library. From any sketch, click the Sketch ➤ Include Library ➤ Manage Libraries menu. This opens the Library Manager. In the filter search box, enter **MySQL**, then choose the connector, and finally click Install. In seconds, the new library is installed and ready for use. Cool, eh? Figure 6-12 shows the Library Manager dialog with the MySQL Connector/Arduino library selected for installation.
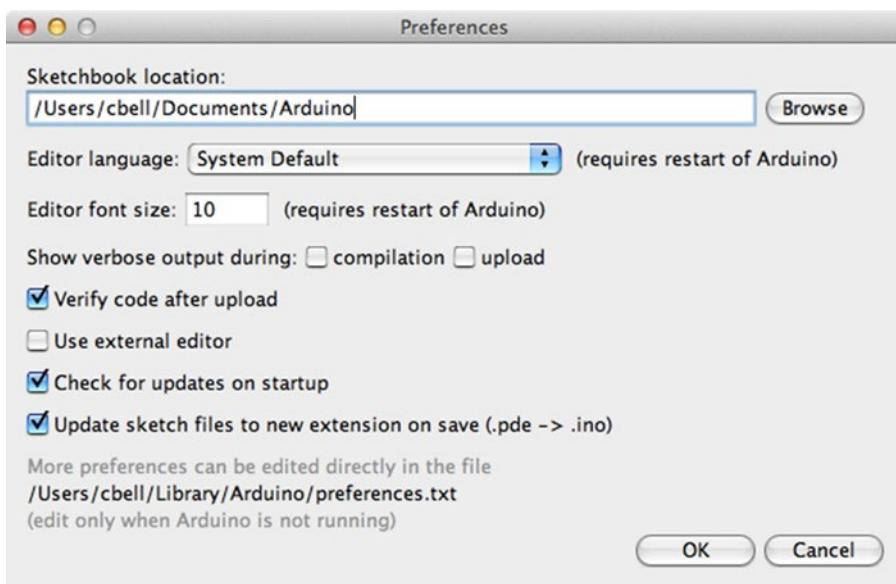
Figure 6-12. *Arduino Library Manager dialog*

If you do not want to use Library Manager or cannot use it because you're using a different IDE or editor, you can download it from the GitHub site (`https://github.com/ChuckBell/MySQL_Connector_Arduino`).

To manually install the connector, begin by navigating to the Connector/Arduino page on GitHub (`https://github.com/ChuckBell/MySQL_Connector_Arduino`). The latest version is always the one available for download. The file is named `MySQL_Connector_Arduino-master.zip`. Look on the right side of the page and click the button to download and save it to your computer. Once it is downloaded, uncompress the file. You will see a new folder in the location where you extracted the file.

You need to copy or move the folder to your `Arduino/Libraries` folder. Place the folder and its contents renamed to `MySQL_Connector_Arduino` in your Arduino library folder. You can find where this is by examining the preferences for the Arduino environment, as shown in Figure 6-13.

Figure 6-13. *Arduino Preferences dialog*

■ **Tip** If you copy a library to your Libraries folder while the Arduino application is running, you must restart it to detect the new library.

Now that you have the Connector/Arduino library installed, you are ready to start writing database-enabled sketches! Before you jump into to the library source code, let's first examine some of the limitations of using the library.

WAIT! I HAVE VERSION 1.0. CAN'T I USE THAT?

If you have already discovered the Connector/Arduino library and have been using version 1.0.4 or older, you will need to upgrade to the newer version to use the examples in this book. This is because a lot of changes were made in the 1.1 version, making it incompatible with the older versions.

But do not despair, because the old version remains in Launchpad and will be left there for some time. Best of all, the new version does not cause conflicts with any of your existing sketches. That is, your existing sketches will not be affected by installing the new library.

However, if you want to use the newest version in your existing sketches, you will have to change a few things. Please see the "Changes from Previous Versions" in the reference manual located in the extras folder of the library source code.

Using Connector/Arduino

Let's begin with a simple sketch designed to insert a single row into a table in MySQL. You are creating a "Hello, world!" sketch (but saved in a database table). All database-enabled sketches share the same common building blocks. These include setting up a database to use, creating a sketch with a specific set of include files, connecting to the database server, and executing queries. This section walks through the basic steps needed to create and execute a database-enabled sketch.

■ **Tip** The library includes a number of examples sketches to get you going quickly. Check out the examples in your quest to master the library. You will find examples of how to connect using WiFi and even how to build complex queries from variables in your sketch.

The first thing you need is a database server! Begin by creating a database and a table to use to store the data. For this experiment, you create a simple table with two columns: a text column (char) to store a message and a TIMESTAMP column to record the date and time the row was saved. I find the TIMESTAMP data type to be an excellent choice for storing sensor data. It is rare that you would not want to know when the sample was taken! Best of all, MySQL makes it easy to use. In fact, you need pass only a token NULL value to the server, and it generates and stores the current timestamp itself.

Listing 6-4 shows a MySQL client (named mysql) session that creates the database and the table and inserts a row into the table manually. The sketch will execute a similar INSERT statement from your Arduino. By issuing a SELECT command, you can see each time the table was updated.

Listing 6-4. Creating the Test Database

```
$ mysql -uroot -psecret
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 102
Server version: 5.6.14-log Source distribution

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE test_arduino;
Query OK, 1 row affected (0.00 sec)

mysql> USE test_arduino;
Database changed
mysql> CREATE TABLE hello (source char(20), event_date timestamp);
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> GRANT ALL ON *.* to 'root'@'%' IDENTIFIED BY 'secret';

mysql> INSERT INTO hello VALUES ('From Laptop', NULL);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM hello;
+-------------+---------------------+
| source      | event_date          |
+-------------+---------------------+
| From Laptop | 2013-02-16 20:40:12 |
+-------------+---------------------+
1 row in set (0.00 sec)

mysql>
```

Starting a New Sketch

It is time to start writing your sketch. Open your Arduino environment, and create a new sketch named hello_mysql. The following sections detail the parts of a typical MySQL database-enabled sketch. You begin with the required include files.

Include Files

To use the Connector/Arduino library, recall that it requires an Ethernet shield and therefore the Ethernet library. The Connector/Arduino library requires the MySQL_Connection library for connecting and the MySQL_Cursor library for running queries. Thus, you must include each of these in order. The following shows all the library header files you need to include at a bare minimum for a MySQL database-enabled sketch. Go ahead and enter these now.

```
#include <Ethernet.h>
#include <MySQL_Connection.h>
#include <MySQL_Cursor.h>
```

Preliminary Setup

With the include files set up, you next must take care of some preliminary declarations. These include declarations for the Ethernet library and Connector/Arduino.

The Ethernet library requires you to set up a MAC address and the IP address of the server. The MAC address is a string of hexadecimal digits and need not be anything special, but it should be unique among the machines on your network. It uses Dynamic Host Control Protocol (DHCP) to get an IP address, DNS, and gateway information. The IP address of the server is defined using the IPAddress class (which stores the value as an array of four integers, just as you would expect).

On the other hand, the Ethernet class also permits you to supply an IP address for the Arduino. If you assign an IP address for the Arduino, it must be unique for the network segment to which it is attached. Be sure to use an IP scanner to make sure your choice of IP address isn't already in use.

The following shows what these statements would look like for a node on a 10.0.1.X network:

```
/* Setup for Ethernet Library */
byte mac_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress server_addr(10, 0, 1, 23);
```

Next, you need to set up some variables for Connector/Arduino. You need to define a reference to the connection class. We dynamically allocate the cursor class later in order to manage memory better. The connection class requires a parameter that is an instance of the `Ethernet.Client` class. If you are using an Arduino Ethernet Shield, you can simply use the `EthernetClient` class. If you are using another library or a WiFi shield, you would use the appropriate client. For example, use WiFiClient for the Arduino WiFi shield.

You also need some strings to use for the data you use in the sketch. At a minimum, these include a string for the user ID, another for the password, and one for the query you use. This last string is optional because you can just use the literal string directly in the query call, but it is good practice to make strings for the query statements. It is also the best way to make queries parameterized for reuse.

The following is an example of the statements needed to complete the declarations for your sketch:

```
/* Setup for the Connector/Arduino */
EthernetClient client;
MySQL_Connection conn((Client *)&client);

char user[] = "root";
char password[] = "secret";
char INSERT_SQL[] = "INSERT INTO test_arduino.hello VALUES ('Hello from Arduino!', NULL)";
```

Notice the INSERT statement. You include a string to indicate that you are running the query from your Arduino. You also include the NULL value so that the server will create the timestamp for the row as shown in the manual execution previously.

Connecting to a MySQL Server

That concludes the preliminaries; let's get some code written! Next, you change the `setup()` method. This is where the code for connecting to the MySQL server should be placed. Recall that this method is called only once each time the Arduino is booted. The following shows the code needed:

```
void setup() {
  Ethernet.begin(mac_addr);
  Serial.begin(115200);
  while (!Serial);
  delay(1000);
  Serial.println("Connecting...");
  if (conn.connect(server_addr, 3306, user, password))
    delay(500);
  else
    Serial.println("Connection failed.");
}
```

The code begins with a call to the Ethernet library to initialize the network connection. Recall that when you use the `Ethernet.begin()` method, passing only the MAC address as shown in the example, it causes the Ethernet library to use DHCP to obtain an IP address. If you want to assign an IP address manually, see the `Ethernet.begin()` method documentation at http://arduino.cc/en/Reference/EthernetBegin.

Next is a call to serial monitor. Although not completely necessary, it is a good idea to include it so you can see the messages written by Connector/Arduino. If you have problems with connecting or running queries, be sure to use the serial monitor so you can see the messages sent by the library.

Now comes a call to the `delay()` method. You issue this wait of one second to ensure that you have time to start the serial monitor and not miss the debug statements. Feel free to experiment with changing this value if you need more time to start the serial monitor.

After the delay, you print a statement to the serial monitor to indicate that you are attempting to connect to the server. Connecting to the server is a single call to the Connector/Arduino library named `connect()`. You pass the IP address of the MySQL database server, the port the server is listening on, and the username and password. If this call passes, the code drops to the next `delay()` method call.

This delay is needed to slow execution before issuing additional MySQL commands. Like the previous delay, depending on your hardware and network latency, you may not need this delay. You should experiment if you have strong feelings against using delays to avoid latency issues. On the other hand, should the connection fail, the code falls through to the print statement to tell you the connection has failed.

Running a Query

Now it is time to run the query. We first instantiate an instance of the `MySQL_Cursor` class and pass in the connection instance. This will dtynamically allocate the class (think code). We then call the `execute()` method and pass in the query we want to run. Since there are no results returned (because we're running an INSERT), we can close the connection and delete the instance. The following shows all these steps in order.

Place this code in the branch that is executed after a successful connection. The following shows the previous conditional statement rewritten to include the method call to run the insert query:

```
if (conn.connect(server_addr, 3306, user, password))
  {
    delay(500);
    /* Write Hello to MySQL table test_arduino.hello */
    // Create an instance of the cursor passing in the connection
    MySQL_Cursor *cur = new MySQL_Cursor(&conn);
    cur->execute(INSERT_SQL);
    delete cur;
  }
  else
    Serial.println("Connection failed.");
}
```

Notice that you simply invoke a method named `execute()` and pass it the query you defined earlier. Yes, it is that easy!

Testing the Sketch

You now have all the code needed to complete the sketch except for the `loop()` method. In this case, you make it an empty method because you are not doing anything repetitive. Listing 6-5 shows the completed sketch.

■ **Tip**   If you are having problems getting the connector working, see the "Troubleshooting Connector/Arduino" section in the MySQL Connector Reference Manual[15] and then return to this project.

---

[15]https://github.com/ChuckBell/MySQL_Connector_Arduino/blob/master/extras/MySQL_Connector_Arduino_Reference_Manual.pdf

Listing 6-5. "Hello, MySQL!" Sketch

```
/**
 * Example: Hello, MySQL!
 *
 * This code module demonstrates how to create a simple database-enabled
 * sketch.
 */
#include <Ethernet.h>
#include <MySQL_Connection.h>
#include <MySQL_Cursor.h>

/* Setup for Ethernet Library */
byte mac_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress server_addr(10, 0, 1, 23);    // The IP address of your database server

/* Setup for the Connector/Arduino */
EthernetClient client;
MySQL_Connection conn((Client *)&client);

char user[] = "root";
char password[] = "secret";
char INSERT_SQL[] = "INSERT INTO test_arduino.hello VALUES ('Hello from Arduino!', NULL)";

void setup() {
  Ethernet.begin(mac_addr);
  Serial.begin(115200);
  while (!Serial);
  Serial.println("Connecting...");
  if (conn.connect(server_addr, 3306, user, password))
  {
    delay(500);
    /* Write Hello, World to MySQL table test_arduino.hello */
    // Initiate the query class instance
    MySQL_Cursor *cur_mem = new MySQL_Cursor(&conn);
    // Execute the query
    cur_mem->execute(INSERT_SQL);
    delete cur_mem;
    Serial.println("Query Success!");
  }
  else
    Serial.println("Connection failed.");
}

void loop() {
}
```
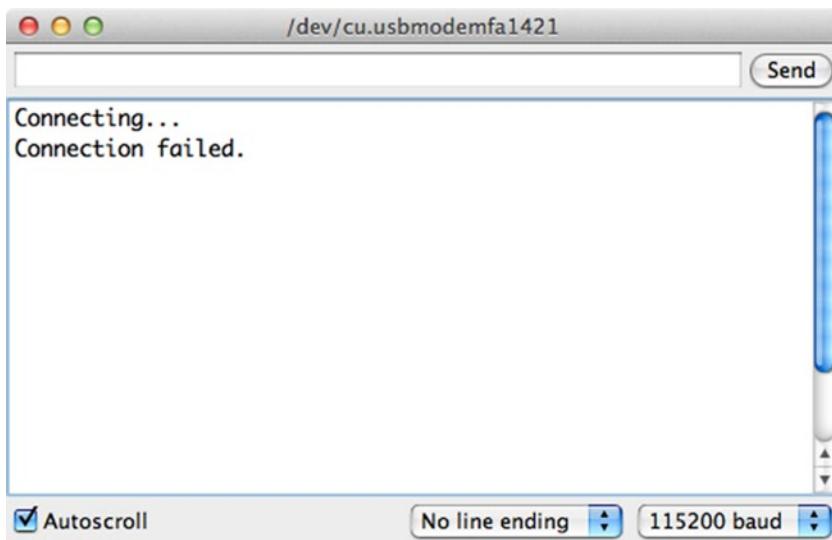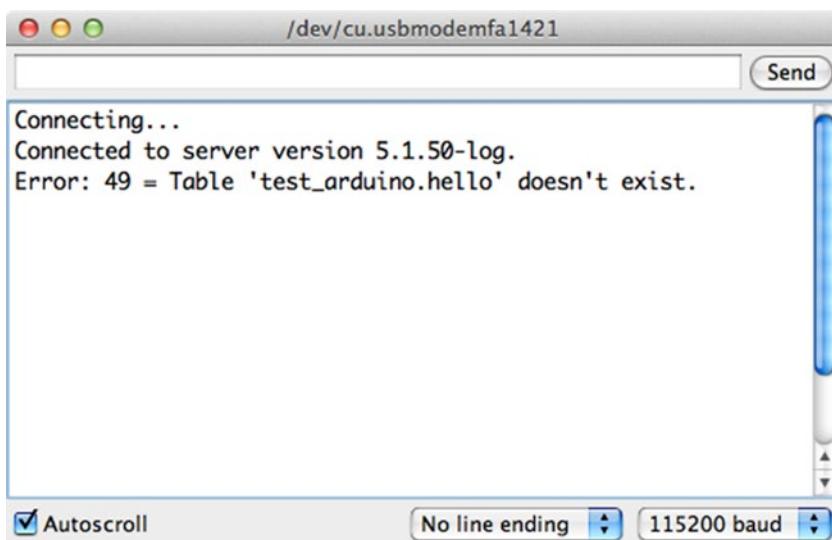
Before you click the button to compile and upload the sketch, let's discuss a couple of errors that could occur. If you have the wrong IP address or the wrong username and password for the MySQL server, you could see a connection failure in the serial monitor like that shown in Figure 6-14.

Figure 6-14. *Failed connection*

If your Arduino connects to the MySQL server but the query fails, you see an error in the serial monitor like the one shown in Figure 6-15.
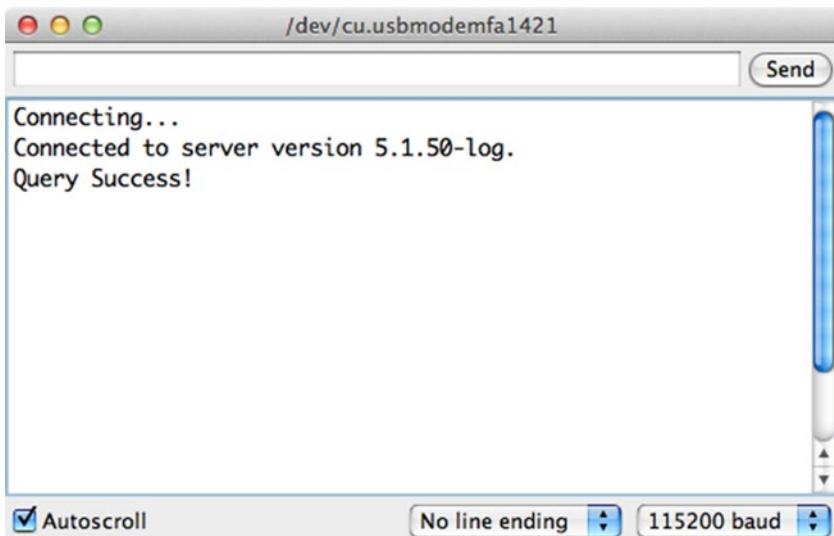
Figure 6-15. *Failed query*

Be sure to double-check the source code and the IP address of your MySQL server as well as the username and password chosen. If you are still encountering problems connecting, see the "Troubleshooting Connector/Arduino" section in the MySQL Connector Reference Manual[16] for a list of things to test to ensure that your MySQL server is configured correctly.

Once you have double-checked the server installation and the information in the sketch, compile and upload the sketch to your Arduino. Then start the serial monitor and observe the process of connecting to the MySQL server. Figure 6-16 shows a completed and successful execution of the code.

■ **Note**   In the examples I am connecting to an older version of MySQL that was installed on my BeagleBone Black board. The sketch will connect to any version of MySQL from 5.0 and later.
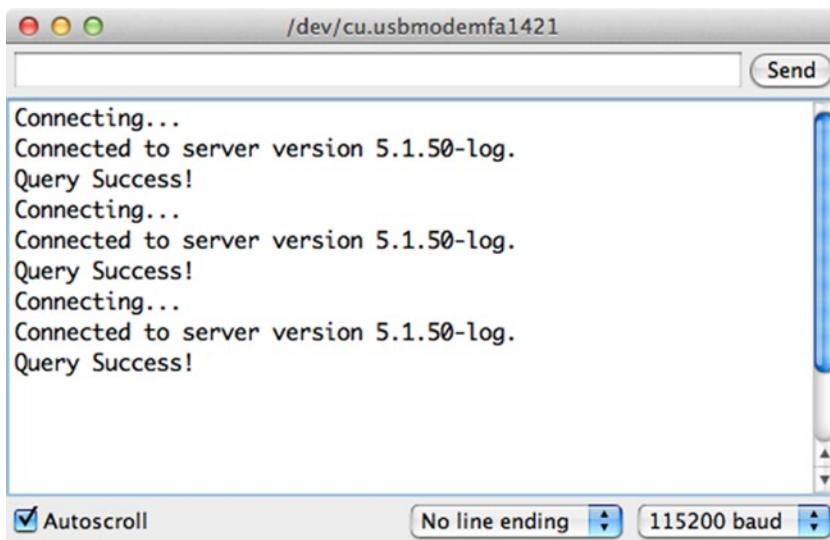
Wow, is that it? Not very interesting, is it? If you see the statements in your serial monitor as shown in Figure 6-16, rest assured that the Arduino has connected to and issued a query to the MySQL server. To check, simply return to the mysql client and issue a select on the table. But first, run the sketch a number of times to issue several inserts in the table.

Figure 6-16. *Correct serial monitor output*

You can do this in two ways. First, you can press RESET on your Arduino. If you leave your serial monitor running, the Arduino presents the messages in order, as shown in Figure 6-17. Second, you can upload the sketch again. In this case, the serial monitor closes, and you have to reopen it. The advantage of this method is you can change the query statement each time, thereby inserting different rows into the database. Go ahead and try that now, and check your database for the changes.

---

[16]https://github.com/ChuckBell/MySQL_Connector_Arduino/blob/master/extras/MySQL_Connector_Arduino_Reference_Manual.pdf

Figure 6-17. *Results of running the sketch several times*

Let's check the results of the test runs. To do so, you connect to the database server with the `mysql` client and issue a SELECT query. Listing 6-6 shows the results of the three runs from the example. Notice the different timestamp for each run. As you can see, I ran it once, then waited a few minutes and ran it again (I used the RESET button on my Arduino Ethernet shield), and then ran it again right away. Very cool, isn't it?

Listing 6-6. Verifying the Connection with the Serial Monitor

```
$ mysql -uroot -psecret
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 33
Server version: 5.6.14-log Source distribution

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select * from test_arduino.hello;
+--------------------+---------------------+
| source             | event_date          |
+--------------------+---------------------+
| From laptop        | 2013-02-19 15:17:38 |
| Hello from Arduino! | 2013-02-19 15:18:12 |
| Hello from Arduino! | 2013-02-19 15:28:39 |
| Hello from Arduino! | 2013-02-19 15:29:16 |
+--------------------+---------------------+
4 rows in set (0.01 sec)

mysql>
```

You can do much more with the connector than shown here. In fact, you can query the database server for lookup data; discover values of variables; create databases, tables, views; and so on. You can do just about anything you want, within reason. Querying the database server for massive rows or large data rows is likely beyond the memory limitations of the Arduino. Fortunately, most of the sketches you will write will be saving data with simple INSERT statements.

■ **Tip** You can find many more examples of how to use the connector in the Connector/Arduino reference manual located in the `extras` folder. The file is named `MySQL_Connector_Arduino_Reference_Manual.pdf`.

Now that you've seen an Arduino connector, let's look at a connector more general purpose and one you can use on your laptop, desktop, low-cost computer, embedded system, and more, Anything that can run Python can write to a MySQL database!

Introducing Connector/Python

The connector for Python from Oracle is a full-featured connector that provides connectivity to the MySQL database server for Python applications and scripts. The latest version is release-2.1.3GA. Unlike the Connector/Arduino, Connector/Python is fully supported and actively maintained by Oracle.

Connector/Python features support for all current MySQL server releases from version 4.1 and newer. It is written to provide automatic data type conversion between Python and MySQL, making building queries and deciphering results easy. It also has support for compression, permits connections via SSL, and supports all MySQL SQL commands. The current version, 2.1.3, is augmented with C libraries to improve performance.

Using Connector/Python in your Python scripts consists of importing the base module, initiating a connection, and executing queries with a cursor, which is similar to Connector/Arduino. That is not surprising since I wrote Connector/Arduino using Connector/Python as a model.

However, unlike Connector/Arduino, Connector/Python has no such memory limitations, allowing you to accomplish a great deal of processing. Indeed, I would move most of my string, date, and mathematical processing to a Python script rather than attempting it on the Arduino.

Before we jump into how we can use Connector/Python to write some MySQL database–enabled applications, let's talk about how to get and install Connector/Python.

PYTHON? ISN'T THAT A SNAKE?

The Python programming language is a high-level language designed to be as close to like reading English as possible while being simple, easy to learn, and powerful. Pythonistas[17] will tell you the designers have indeed met these goals.

Python does not require a compilation step prior to being used. Rather, Python applications (whose file names end in .py) are interpreted on the fly. This is very powerful, but unless you use a Python integrated development environment (IDE) that contains an automatic syntax checker, some syntax errors will not be discovered until the application is executed. Fortunately, Python provides a robust exception-handling mechanism that will communicate what has gone wrong.

---

[17]Python experts often refer to themselves using this term. It is reserved for the most avid and experienced Python programmers.

If you have never used Python or you would like to know more about it, the following are few good books that introduce the language. A host of resources are also available on the Internet, including the Python documentation pages at `www.python.org/doc/`.

- *Programming the Raspberry Pi* by Simon Monk (McGraw-Hill, 2013)

- *Beginning Python from Novice to Professional, 2nd Edition*, by Magnus Lie Hetland (Apress, 2008)

- *Python Cookbook* by David Beazley and Brian K. Jones (O'Reilly Media, 2013)

Interestingly, Python was named after the British comedy troupe Monty Python and not the reptile. As you learn Python, you may encounter campy references to Monty Python episodes. Having a fondness for Monty Python, I find these references entertaining. Of course, your mileage may vary.

Installing Connector/Python

Downloading is the same process as you discovered for the server. You can download Connector/Python from Oracle's MySQL web site (`http://dev.mysql.com/downloads/connector/python/`). The page will automatically detect your platform and show the available downloads for your platform. You may see several choices. Be sure to choose the one that matches your configuration.

Installing on Desktop/Laptop Platforms

Since most platforms come with Python installed, you may not need to do anything to prepare your system; just download the installer and install it. You can find the latest Python installers on the product download page (`http://dev.mysql.com/downloads/connector/python/`). Note that Connector/Python requires either Python 2.7 (recommended) or Python 3.3.

Figure 6-18 shows a typical download page for the Ubuntu platform. Use the drop-down box to select a different platform if yours is not listed. However, if you use a platform such as Windows that does not include Python, you should install Python first.

Figure 6-18. *Connector/Python download page*

As I mentioned, the current version of Connector/Python is 2.1.3, but most versions 2.0 or later should be fine for your solutions. The newest releases add high availability features for use with the enterprise-level high availability solution named MySQL Fabric. Any release of 2.0 or 2.1 will be fine for use with IOT solutions.

■ **Tip** See the online reference manual for specific notes about installing on some platforms (http://dev.mysql.com/doc/connector-python/en/connector-python-installation.html).

Installing on Low-Cost Platforms

Installing Connector/Python on the smaller platforms is a bit more involved but not overly so. In short, you install Connector/Python by using the packaging mechanism for the platform (apt-get, opkg, and so on). The following shows how to install Connector/Python on the Raspberry Pi using Raspbian Jessie. Other platforms are similar.

```
$ sudo pip3 install mysql-connector-python --allow-external mysql-connector-python
Downloading/unpacking mysql-connector-python
  mysql-connector-python an externally hosted file and may be unreliable
  Downloading mysql-connector-python-2.0.4.zip (277kB): 277kB downloaded
  Running setup.py (path:/tmp/pip-build-mm9szi9x/mysql-connector-python/setup.py) egg_info
  for package mysql-connector-python

Installing collected packages: mysql-connector-python
  Running setup.py install for mysql-connector-python

Successfully installed mysql-connector-python
Cleaning up...
```

Notice I used the Python package manager (from PyPi) to get and install the connector. This installs an older version of the connector, but it is fully functional and will meet your IOT solution needs.

Checking the Installation

Once Connector/Python is installed, you can verify it is working with the following short example. Begin by entering the command python. This will open an interactive prompt that permits you to enter one line of Python code at a time and execute it; it's a Python command-line interpreter and useful in testing small snippets of code. Just enter the following lines as shown in the example:

```
$ python
Python 2.7.6 (default, Mar  4 2014, 16:53:21)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.2.79)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import mysql.connector
>>> print mysql.connector.__version__
2.1.3
>>> quit()
```

What you should see is the version of Connector/Python printed. If you see any errors about not finding the connector, be sure to check your installation to ensure it worked. Once you can successfully access Connector/Python, you're ready to move on to some examples.

■ **Tip**   If you have multiple versions of Python installed and installed Connector/Python under a different Python version than the default, use the version-specific Python executable. For example, if you installed Connector/Python under Python3 but Python2.7 is the default, use the command python3 to start the interpreter. Otherwise, you may see errors when doing the import.

Using Connector/Python

Let's start with a simple example where we connect to the MySQL server and get a list of databases. In this case, we start by importing the Connector/Python connector class and then call the connect() method to connect to the server. To keep things tidy, we use a dictionary to store the connection information. Be sure your MySQL server is running and you change the following example to match your setup. For example, provide the correct password and hostname for the server.

```
$ python
Python 2.7.6 (default, Mar  4 2014, 16:53:21)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.2.79)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import mysql.connector
>>> server = {
...     'user': 'root',
...     'password': <secret>,
...     'host': '127.0.0.1',
...     'database': 'employees',
...     'raise_on_warnings': True,
... }
>>> cnx = mysql.connector.connect(**server)
```

If you get errors at this point, go back and check your connection parameters. It is most likely that your MySQL server is unreachable, it is down, or you have the wrong credentials specified. If you get no response from the interactive interpreter, that's OK—you've connected and you are ready to go!

Now, leave the interpreter running and add the following statements. Here we will open a cursor object for executing queries and retrieving rows. I wrote the loop in a simplistic way to show you how to loop through all available rows (there are several other, valid ways to write that bit).

```
>>> cur = cnx.cursor()
>>> cur.execute("SHOW DATABASES")
>>> rows = cur.fetchall()
>>> for row in rows:
...     print row
...
```

When you get the `...` (which is a prompt), press Enter and observe the results as shown here. Your list may be slightly different depending on what databases are on your server.

```
(u'arduino',)
(u'employees',)
(u'library',)
(u'mysql',)
(u'performance_schema',)
(u'plant_monitoring',)
(u'test',)
(u'test_arduino',)
(u'world',)
(u'world_innodb',)
```

But we're not done. There are two more steps needed. We need to close the cursor and connection and then exit the interpreter.

```
>>> cur.close()
True
>>> cnx.close()
>>> quit()
```

Congratulations! You've written your first MySQL-enabled Python script. Now that we know the basics, let's move on to more powerful scripts using a source file instead of the interactive interpreter.

Most Python scripts (applications) are built using a file named <something>.py and executed from the command line as follows. We will use this method to execute the following examples. Thus, for each example, you should open a file and enter the text as shown.

```
$ python my_script.py
```

Now let's see how we can insert some data in a table. In this case, we simply want to read data from a file and insert it into a table. I'll let you use your imagination for how you could change the file to reading sensors. In fact, I will show you how to do this in a later chapter.

■ **Note**   Refer to Chapter 5 for the table layout. Be sure to empty the table if you performed the examples from Chapter 5 so you can avoid key violations when running this example.

Open your favorite text editor and enter the code shown in Listing 6-7. Save the file with the name simple_insert.py.

Listing 6-7. Inserting Data with Connector/Python

```python
import mysql.connector
server = {
  'user': 'root',
  'password': <secret>,
  'host': '127.0.0.1',
  'database': 'employees',
  'raise_on_warnings': True,
}
cnx = mysql.connector.connect(**server)
cur = cnx.cursor()
# read rows from a file for inserting into plant_monitor table
f = open("plants_data.txt")
lines = f.readlines()
f.close()
# now insert the data
for line in lines:
    cols = line.strip('\n').split(",") # comma-separated row
    query = "INSERT INTO plant_monitoring.plants (name, location, climate)" \
            " VALUES ('{0}','{1}',{2});".format(cols[0], cols[1], cols[2])
    print query
    cur.execute(query)
cnx.commit()
cur.close()
cnx.close()
```

Here we see the same startup code as the previous example only this time we're reading values from a file and performing an INSERT SQL statement on each. Take a moment to study the code and how it works. Note that it uses string substitution.

Notice the line in bold. This command is necessary to ensure the rows are written to the table. More specifically, since I used a transactional storage engine and my server is set up for transactions, none of the data is written until I explicitly commit the changes. Your server may be set up differently, but it does not hurt to add this command here. You could add it inside the loop too, but it is best to push your commits out to the latest code block, and in this case it is outside the loop.

The file we are reading has only a few rows and is a mockup of the plant-monitoring system example from Chapter 5. Listing 6-8 shows the file contents. Note that I labeled it plants_data.txt. If you change the file name, be sure to change the code accordingly.

Listing 6-8. Sample Data

```
Jerusalem Cherry,deck,2
Moses in the Cradle,patio,2
Peace Lilly,porch,1
Thanksgiving Cactus,porch,1
African Violet,porch,1
```

To run the script, issue the following command from the folder where you stored the file. Be sure to put the data file in the same folder first. I show the results of running the script.

```
$ python ./simple_insert.py
INSERT INTO plant_monitoring.plants (name, location, climate) VALUES ('Jerusalem
Cherry','deck',2);
INSERT INTO plant_monitoring.plants (name, location, climate) VALUES ('Moses in the
Cradle','patio',2);
INSERT INTO plant_monitoring.plants (name, location, climate) VALUES ('Peace
Lilly','porch',1);
INSERT INTO plant_monitoring.plants (name, location, climate) VALUES ('Thanksgiving
Cactus','porch',1);
INSERT INTO plant_monitoring.plants (name, location, climate) VALUES ('African
Violet','porch',1);
```

Now let's check our table. If we started with an empty table, we should see the following:

```
mysql> SELECT * FROM plant_monitoring.plants;
+----+---------------------+----------+---------+
| id | name                | location | climate |
+----+---------------------+----------+---------+
| 30 | Jerusalem Cherry    | deck     | outside |
| 31 | Moses in the Cradle | patio    | outside |
| 32 | Peace Lilly         | porch    | inside  |
| 33 | Thanksgiving Cactus | porch    | inside  |
| 34 | African Violet      | porch    | inside  |
+----+---------------------+----------+---------+
5 rows in set (0.00 sec)
```

You can do much more with the connector than shown here. In fact, you can do just about anything you want. Typically, I use Python scripts for performing complex operations on my databases and database servers. For example, I may write a script to set up all of my databases, tables, functions, and so on, so that I can reload a test or start an experiment on any server I want; I just supply different connection parameters and run it.

MYSQL UTILITIES: PYTHON-BASED DATABASE ADMINISTRATION

If you are a Python developer and find yourself working more and more with MySQL, especially if you have found yourself in the role of administrator, you may want to look at MySQL Utilities from Oracle. MySQL Utilities is a set of Python scripts and a Python library for managing MySQL servers. You can do all manner of things from copying user permissions to cloning servers to discovering differences between two databases. See `http://dev.mysql.com/downloads/utilities/` for more details.

You may also want to write complex Python scripts for manipulating your data from your sensors or IOT data collectors. That is, you could use a Raspberry Pi as a data aggregator for preparing data for storage. You can even use Python applications to read sensors directly from the Raspberry Pi as I demonstrated in my book *Beginning Sensor Networks with Arduino and Raspberry Pi* (Apress, 2014).

For more complex examples including executing transactions, creating tables, and running complex queries, see the coding examples section in the Connector/Python online reference manual (`http://dev.mysql.com/doc/connector-python/en/connector-python-examples.html`).

Summary

This chapter introduced MySQL and gave you a crash course on how to set up a Raspberry Pi, install MySQL, and use it. You also learned how to write data to your database server using an Arduino sketch and a Python program on another machine (Raspberry Pi, BeagleBone Black, pcDuino, and so on).

Although it does not have nearly the sophistication of a high availability, five-nines uptime (99.999 percent) database server, the low-cost Raspberry Pi with an attached USB hard drive makes for a very small-footprint database server that you can put just about anywhere. This is great because IOT solutions, by nature and often by necessity, need to be small and low cost. Having to build an expensive database server is not usually the level of investment desired.

In the next chapter, we will explore an advanced topic: high availability. More specifically, we will see how to make our database server more reliable by providing redundancy as well as an ability to separate reads (SELECT) and writes (INSERT, UPDATE, DELETE) across multiple database nodes.

■ ■ ■

High Availability IOT Solutions

You may be wondering what high availability has to do with IOT solutions. That is, you may have thought high availability is only for large enterprises, is far too costly, or is extremely complicated. While it is true that high availability solutions can be taken to these extremes[1] and that reaching 100 percent uptime is difficult and costly,[2] it is also true that high availability is largely misunderstood. It is not that complicated in concept nor is it that difficult to achieve at lower levels of reliability using a more modest investment.

It is also true that high availability can mean different things to different people. Indeed, if you read popular trade journals, books, blogs, and so on, devoted to high availability, chances are you will become confused by the different viewpoints. This is because there are many ways you can implement high availability, each of which may solve one particular aspect or conform to certain architectures. There simply is no single high availability implementation (solution) that meets every possible need.

However, I am not suggesting that high availability is something you can just switch on. As you will see, it does require some work, but depending on your IOT solution, it may make the difference between a solution that works well in a small scale but fails when expanded or when put into production in the field. The challenge for developers is to know what tools are available and how to employ them to achieve one or more goals of high availability.

This chapter introduces high availability as it relates to IOT solutions, built from commodity hardware, that use MySQL to store and retrieve data. Thus, the focus will be on leveraging the free tools and features in MySQL to achieve high availability. You will see several options available to you for adding high availability MySQL options to your solution as well as examples of how to set up MySQL for high availability. Let's begin by explaining high availability.

What Is High Availability?

High availability is easiest to understand if you consider it loosely synonymous with reliability—making the solution as accessible as possible and tolerant to failures either planned or unplanned for an agreed upon period of time. That is, it's how much users can expect the system to be operational. The more reliable the system and thus the longer it is operational equates to a higher level of availability.

High availability can be accomplished in many ways, resulting in different levels of availability. The levels can be expressed as goals to achieving some higher state of reliability. Essentially, you use techniques and tools to boost reliability and make it possible for the solution to keep running and the data to be available as long as possible (also called *uptime*). Uptime is represented as a ratio or percentage of the amount of time the solution is operational.

---

[1]Large organizations know all too well the stratospheric costs of implementing high availability.
[2]And therefore impossible.

RELIABILITY VS. HIGH AVAILABILITY: WHAT IS THE DIFFERENCE?

Reliability is a measure of how operational a solution is over time, which covers one of the major goals for high availability. Indeed, you could say that the ultimate level of reliability—the solution is always operational—is the definition of high availability. Thus, to make your solution a high availability solution, you should focus on improving reliability.

You can achieve high availability by practicing the following engineering principles:

- *Eliminate single points of failure*: Design your solution so that there are as few components as possible that, should they fail, render the solution unusable.

- *Add recovery through redundancy*: Design your solution to permit multiple, active redundant mechanisms to allow rapid recovery from failures.

- *Implement fault tolerance*: Design your solution to actively detect failures and automatically recover by switching to a redundant or alternative mechanism.

These principles are building blocks or steps to take to reach higher levels of reliability and thus high availability. Even if you do not need to achieve maximum high availability (the solution is up nearly all of the time), by implementing these principles you will make your solution more reliable at the least, which is a good goal to achieve.

These principles may not seem like reasonable requirements for a simple IOT project that you build yourself, but it can make a huge difference for larger IOT solutions. For example, suppose you wanted to take a plant-monitoring solution like the one suggested in this book and build it into a solution to manage plants in hundreds of greenhouses for a large nursery. Some may tell you it isn't possible or that there are far too many problems to solve to get all the data in one place. However, that would be short-sighted in the least.

On the contrary, this goal is achievable in concept as well as implementation. You can indeed set up thousands of plant monitors[3] and link them all together to be monitored from an application via the Internet. The problem comes when you expect all the hardware to work all the time and never fail. That is, what do you do if a major component such as the database server goes offline through either failure or required maintenance? How do you recover from that?

Moreover, how do you keep your solution going should some of the intermediate nodes fail? For IOT solutions, this includes the key components such as the application, database server, web server, and, depending on the critical nature of the solution, even the data collectors or intermediate nodes.

You can achieve several characteristics or goals of high availability in your solution by building it with specific tools or techniques. Table 7-1 lists a number of high availability goals and possible implementations.

Table 7-1. *High Availability Goals for IOT Solutions*

Goal	Technique	Tools
Recover from storage media failure	Recovery	Backup and restore tools
Quickly recover from database failure	Redundancy	Multiple copies of the database
Improve performance	Scaling	Splitting writers and readers
Have no loss of data collection	Fault tolerance	Caching data and using redundant nodes

[3]Some sensors would be per tray or even per deck since greenhouses typically have many plants sharing the same soil.

As you can see, there are several concepts and corresponding tools or techniques that you can employ to achieve different goals or levels of high availability. Notice too that some goals have overlapping solutions. Keep in mind this list does not cover every high availability goal; rather, it lists those that provide high availability capabilities that are relatively easy to achieve with a minimal amount of investment, in other words, goals you can use to achieve high availability in your IOT solutions with known solutions and techniques. I discuss these goals in more detail in the next section.

SO, WHAT IS FIVE NINES?

You may have heard or read about a concept called *five nines*, or 99.999 percent of a year uptime. A five nine solution therefore permits, at most, only 5.26 minutes of downtime per year. But five nines is just one class or rating regarding reliability that includes other categories, each related to the percentage of uptime or reliability. See https://en.wikipedia.org/wiki/High_availability#Percentage_calculation for more information about the available classes.

High Availability Options for IOT Solutions with MySQL

Now that you understand the goals or requirements that high availability (HA) can solve, let's now discuss some of the options for implementing HA in your IOT solutions. Since we're placing the data in a database server, we will concentrate on techniques and tools for MySQL. However, there are some things you can do outside of the database server to help achieve better reliability.

CAN MYSQL REALLY REACH HIGH AVAILABILITY?

Not only can you reach high availability with MySQL, there are many options for achieving high availability with MySQL, some from third-party vendors as well as several tools by Oracle. Even MySQL itself is designed with the basic building blocks for high availability. However, the features of MySQL as well as the tools and solutions for high availability allow you to tailor MySQL to provide as much reliability as you need. For more information and an in-depth look at the details of high availability solutions for MySQL, see *MySQL High Availability* by Bell, Kindahl, and Thalmann (O'Reilly, 2014).

The following sections discuss four options for implementing goals of high availability. By implementing all of these, you will achieve a level of high availability in your IOT solution. How much you achieve depends on not only how you implement these options but also how well you meet your goals for reliability.

Recovery

The easiest implementation of reliability you can achieve is the ability to recover from failures. This could be a failure in a component, node, database server, or any other part of the solution. Recovery therefore is how to get the solution back to operation in as little time and cost as possible.

However, it may not be possible to recover from all types of failure. For example, if one or more of your data collectors fail, recovery may require replacing the hardware and loss of data during the outage. For other types of failure, recovery options may permit a faster method of returning to operation. Furthermore,

some components are more important and must be recoverable, and thus your efforts should be to protect those more important components, the database being chief among them.

For instance, if your data becomes corrupt or is lost because of hardware failure, you need to have a way to recover that data with as little loss as possible. One way to achieve that is by keeping frequent backup copies of the data that can later be restored to recover the data from loss.

Many tomes[4] have been written about various strategies for backing up and restoring your data. Rather than attempt to explain every nuance, technique, and best practice, I refer you to the many texts available. For this chapter and the solutions available for MySQL, it is sufficient to understand there are two types of backup methods (logical and physical), each with its own merits.

A logical backup makes a copy of the data by traversing the data, making copies of the data row by row, and typically translating the data from its binary form to SQL statements. The advantage of a logical backup is the data is human readable and can even be used to make alterations or corrections to the data prior to restoring it. The downside is logical backups tend to be slow for larger amounts of data and can take more space to store than the actual data (depending on data types, number of indexes, and so on).

A physical backup makes a binary copy of the data from the disk storage layer. The backup is typically application specific; you must use the same application that made the backup to restore it. The advantage is the backup is much faster and smaller in size. Plus, applications that perform physical backups have advanced features such as incremental backups (only the data that has changed since the last backup) and other advanced features. For small solutions, a logical backup may be more than sufficient, but as your solution (your data) grows, you may want to consider a physical backup solution.

Redundancy

One of the more challenging implementations of reliability is redundancy—having two or more components serving the same role in the system. A goal for redundancy may be simply having a component in place in case you need to replace the primary. This could be a hot standby where the component is actively participating in parallel with the primary where your system automatically switches to the redundant component when a failure is detected. The most common target for redundancy is the database server. MySQL excels in this area with a feature called *replication*.

MySQL replication is not difficult to set up for the most basic use cases, which are hot standby and backup. For these, you set up a second database server that gets a copy of all changes made on the original server. The original server is called the *master* or the *primary*, and the second server is called the *slave* or *secondary*. MySQL replication is such a large topic that I've devoted a section to it later in this chapter.

Redundancy can also be implemented in your IOT network by including redundant nodes, including redundant data collectors, or even using multiple sensors in case one fails. It is less likely that you would do this, but it is indeed possible, and I've seen some examples of redundant data nodes. For example, you could use two microcontrollers as data nodes; one is a primary that connects to your data collectors retrieving data. Meanwhile, the second microcontroller periodically exchanges a message with the first microcontroller (called a *handshake*). Thus, you would implement a rudimentary signal/return method for when the second microcontroller fails to respond.

Another possible redundancy measure is writing the code on your data aggregators (nodes that write data to the database) to detect when the database server no longer responds (the connection fails) and to switch to writing the data to a local log.[5] The code would check the database server periodically by trying to reconnect. Once it reconnects, it reads the log and inserts the data into the database.

---

[4]Some of the earlier works cover so much material, they can be used as boat anchors, ballasts, and even building materials. If you know what green bar is, I've seen a set of backup documents that filled a box for green bar paper.
[5]Some call this *logging*, while others may call it *caching*.

This is a good strategy and one of the most common found in DIY IOT solutions. However, there is a downside. If you use date and time fields such as a timestamp on the database server, the date and time the recovered data is saved will be in error. In this case, you would have to save the correct date and time when the data is written to the log.

There are other implementations of redundancy you could implement in your IOT solution. You could implement a redundant power option (e.g., solar, battery), use multiple sensors to guard against failure or multiple communication protocols in case one fails (use XBee modules in case of WiFi failures), and more. There isn't really any reason you cannot build redundancy into your solution. However, only you, the designer, will know which nodes are the most critical and therefore which ones you want to have duplicates of in case of failure.

The sophistication of the redundant mechanism is something you control and depends on how much you want to put into it. In fact, the level of sophistication of the redundancy is associated with the amount of work or expense of the implementation.

For example, you could use a spare component that can be manually activated when the original fails, which is slow and requires manual intervention. Or you can use an active component that can be used in place of the primary, which still requires manual intervention but is faster to recover. Or you can write your code to automatically detect the failure and switch to the secondary, which is the best (fastest) but requires more programming and thus more work (potentially a lot more).

Thus, you can tailor your redundancy to meet your needs or abilities. You could start with simple offline spares and add greater sophistication as your solution evolves.

Scaling

Another reliability implementation has to do with performance. In this case, you want to minimize the time it takes to store and retrieve data. MySQL replication is an excellent way to implement scalability. You do this by designing your solution to write (save) data to the master (primary) and read the data from the slave (secondary). As the application grows, you can add additional slaves to help minimize the time to read data. Having additional slaves allows your application to run more than one instance or even multiple connections simultaneously (one per slave at a minimum). Thus, scalability builds upon the redundancy features in MySQL.

By splitting the writes and reads, you relieve the master of the burden of having to execute many statements. Given most applications have many more reads than writes, it makes sense to devote a different server (or several) to providing data from reading and leaving the writes to the one master server.

Scalability may not be the most urgent for most IOT solutions, but it can be a good way to improve performance for larger solutions or solutions with a lot of data. I will show an example of scalability using MySQL in the next chapter, but the concepts for how you set up MySQL replication are the same as creating a hot standby. The difference is building into your application the ability to split the writes and reads across the replication servers.

Of course, there are other ways to improve performance that do not require implementing MySQL replication, but you may not achieve much benefit in the longer run. You are more likely to improve performance with faster components for cases where the data collectors are slower than expectations.

Fault Tolerance

The last implementation of reliability and indeed what separates most high availability solutions with regard to uptime is fault tolerance, which is the ability to detect failures and recover from the event. Fault tolerance is achieved by leveraging recovery and redundancy and adding the detection mechanism and active switchover.

For example, if a data collector goes offline and the data being collected is critical, your solution should detect that the data collector is offline and switch to a redundant data collector. Thus, having redundant data collectors is required to implement this example. I will show an example of redundant data collectors in the next chapter.

You can also implement fault tolerance at the database. Once again, we build on the use of MySQL replication to achieve the switch. That is, when the master goes down, we use the replication commands in MySQL to switch the role of master to one of the slaves. There are two types of the master role change when working with MySQL: switchover, which is switching the role of master to a slave when the master is still operational, and failover, which is selecting a slave to take on the role of master when the master is no longer operational. That is, switchover is intentional, and failover is a reactive event.

Oracle provides a couple of tools to help you set up automatic failover. You can use MySQL Utilities (`mysqlfailover`) to monitor your master and switch to a slave when the master goes offline. For larger solutions with many servers, you can use MySQL Fabric to manage an entire server farm, performing failover automatically as well as other more sophisticated high availability operations. There is also MySQL Router, which is a connection router for MySQL that allows you to set up a specific set of servers to be used by the router such that the router automatically switches to another server should the current server go offline (become unreachable). You can find all of these products on the MySQL download page (`http://dev.mysql.com/downloads/`). All of them are open source products.

Now that we've discussed some of the implementations high availability can take for IOT solutions, let's look at some demonstrations of the techniques for implementing or setting up the concept.

High Availability Techniques

You can implement high availability concepts in many ways. There are so many ways that it would require an entire book to explain even a portion of the more common techniques. Recall that it is often not necessary to try to implement every technique because you either simply don't need it or the cost of implementation is greater than the benefits. With this in mind, this section introduces some of the most common techniques that you will want to consider for implementation in your IOT solutions starting with backup and recovery.

Backup and Recovery

Recall that the simplest concept of reliability is backup/recovery. To achieve success, you must plan, execute, and audit data backup and restore operations. The one aspect that is most often overlooked is auditing. A good backup and restore should be reliable. That is, the output of the backup should be checked or audited to ensure it is complete and can be restored successfully. Thus, you should test your backups by restoring them on a test machine to ensure they are viable should you need them.

This section describes some of the concepts and tools you will need to make backups of your MySQL data. I discuss the concepts in some detail for those who may not be familiar with backup and recovery systems and the available solutions for MySQL. Let's begin by defining the goals for backup and restore.

A backup operation must make an exact copy of the data. Furthermore, the backup copy must be consistent. That is, the backup contains only transactions committed prior to the start of the copy, not partial or uncommitted data. A restore operation must replace the data on the system with data that is in the backup archive so that the resulting data is identical to that in the archive.

Unfortunately, few backup solutions meet all of these criteria for backup and restore. Those that do are often proprietary and are expensive and difficult to maintain. The next section introduces some economical options to back up and restore your MySQL data.

Fortunately, there are several products that you can use to make backups of your MySQL data both logical and physical. In fact, Oracle provides several products including three open source options as well as a paid option. Table 7-2 lists the options available from Oracle for making backups of your data.

Table 7-2. *Backup Options for MySQL*

Tool	Type	License	URL	Notes
mysqldump	Logical	Open source	http://dev.mysql.com/doc/refman/5.7/en/mysqldump.html	Included with the server install
mysqlpump	Logical	Open source	http://dev.mysql.com/doc/refman/5.7/en/mysqlpump.html	Included with the server install
mysqldbexport and mysqldbimport	Logical	Open source	http://dev.mysql.com/doc/index-utils-fabric.html	Included in MySQL Utilities
MySQL Enterprise Backup	Physical	Fee-based	http://dev.mysql.com/doc/index-enterprise.html	Included in MySQL Enterprise subscriptions
File Copy	Physical	Free		Operating system tool

Notice I include the name of the tool, the type of backup produced, a license, and a link to find out more about the tool and make the best solution for your needs. I discuss each of these in more detail in the following sections.

The mysqldump Client

A popular logical backup tool is the mysqldump client application. It has been part of the MySQL server for some time and installed automatically when you install the server. The client creates a set of SQL statements that re-create the databases when you rerun them. For example, the output contains all the CREATE statements needed to create the databases and the tables as well as all the INSERT statements needed to re-create the data. You use the mysql client utility to read the file to restore it.

This form of backup can be useful in changing or correcting data. Simply back up your database, edit the resulting statement, and then rerun the statements to effect the changes. One possible use of this technique is to correct data values that need to be categorized (the transformation criteria was changed) or perhaps to change the calibration of the data.

However, recall that logical backups and in this case a file containing SQL statements can be slow to back up and slower to restore. This is because the database server has to read each statement and execute them. Thus, you are forcing the database server to redo the work done to save the data originally.

You can use mysqldump to back up all your databases, a specific subset of databases, or even particular tables within a given database. Listing 7-1 shows an example of backing up a specific table for a specific database.

Listing 7-1. Using mysqldump to Back Up a Table

```
$ mysqldump -uroot --password  plant_monitoring plants
Enter password:
-- MySQL dump 10.13  Distrib 5.7.8-rc, for osx10.8 (x86_64)
--
-- Host: localhost    Database: plant_monitoring
-- ------------------------------------------
-- Server version         5.7.8-rc-log
```

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;


--
-- Table structure for table `plants`
--

DROP TABLE IF EXISTS `plants`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `plants` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` char(50) DEFAULT NULL,
  `location` char(30) DEFAULT NULL,
  `climate` enum('inside','outside') DEFAULT 'inside',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;


--
-- Dumping data for table `plants`
--

LOCK TABLES `plants` WRITE;
/*!40000 ALTER TABLE `plants` DISABLE KEYS */;
INSERT INTO `plants` VALUES (1,'Jerusalem Cherry','deck','outside'),(2,'Moses in the
Cradle','patio','outside'),(3,'Peace Lilly','porch','inside'),(4,'Thanksgiving Cactus',
'porch','inside'),(5,'African Violet','porch','inside');
/*!40000 ALTER TABLE `plants` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2015-11-23 20:25:07
```

Notice the verbosity of the output. As you can see, the client captures a great deal of information about the server, the database, and the tables. Notice also that the default is to use bulk insert statements. And, yes, this file can be piped into a `mysql` client and executed. There are many options that allow you to control how the client works. If creating a backup in the form of SQL statements sounds like the best option for you, see the online MySQL reference manual on `mysqldump` (http://dev.mysql.com/doc/refman/5.7/en/mysqldump.html).

The mysqlpump Client Utility

The `mysqlpump` client is a newer variant of the `mysqldump` client utility. It has been completely redesigned for greater speed for both backup and restore. There are additional options for managing the way the backup is created as well as advanced features not available in the older client. However, in concept it works the same way as `mysqldump`. Listing 7-2 shows an example of the output of `mysqlpump`.

Listing 7-2. Using the `mysqlpump` Client Utility

```
$ mysqlpump -uroot -p plant_monitoring --skip-definer
Enter password:
-- Dump created by MySQL pump utility, version: 5.7.8-rc, osx10.8 (x86_64)
-- Dump start time: Mon Nov 23 20:38:01 2015
-- Server version: 5.7.8

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_TIME_ZONE=@@TIME_ZONE;
SET TIME_ZONE='+00:00';
SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT;
SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS;
SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION;
SET NAMES utf8;
CREATE DATABASE /*!32312 IF NOT EXISTS*/ `plant_monitoring` /*!40100 DEFAULT CHARACTER SET
latin1 */;
CREATE TABLE `plant_monitoring`.`plants` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`name` char(50) DEFAULT NULL,
`location` char(30) DEFAULT NULL,
`climate` enum('inside','outside') DEFAULT 'inside',
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1
;
INSERT INTO `plant_monitoring`.`plants` VALUES (1,"Jerusalem Cherry","deck","outside"),
(2,"Moses in the Cradle","patio","outside"),(3,"Peace Lilly","porch","inside"),
(4,"Thanksgiving Cactus","porch","inside"),(5,"African Violet","porch","inside");
Dump progress: 0/1 tables, 5/0 rows
DELIMITER //
CREATE FUNCTION `plant_monitoring`.`max_samples_today`(in_id int) RETURNS int(11)
    READS SQL DATA
    DETERMINISTIC
```

```
BEGIN
  DECLARE num_samples int;
  SELECT COUNT(*) into num_samples FROM plant_monitoring.readings
  WHERE DATE(event_time) = CURRENT_DATE() AND readings.id = in_id;
  RETURN num_samples;
END//
DELIMITER ;

;
SET TIME_ZONE=@OLD_TIME_ZONE;
SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT;
SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS;
SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
-- Dump end time: Mon Nov 23 20:38:01 2015
Dump completed in 549 milliseconds
```

As you can see, the output is similar. Like `mysqldump`, there are many options that allow you to control how the client works. If creating a backup in the form of SQL statements sounds like the best option for you, see the online MySQL reference manual on `mysqlpump` (http://dev.mysql.com/doc/refman/5.7/en/mysqlpump.html).

MySQL Utilities Database Export and Import

MySQL Utilities is a set of utilities written in Python designed to provide a solution for MySQL that the growing culture of development operations (DevOps[6]) can use to automate many repetitive tasks. More specifically, the utilities help you manage MySQL effectively. Many utilities are available, but this section introduces two utilities that you can use to help back up and restore your data. They may be helpful when you need to make a copy of your data either for transformation (bulk or targeted changes to your data) or to make a human-readable copy of the data.

The first utility is `mysqldbexport`. This utility permits you to read your databases (a selected list or all databases) and produces output in one of several formats, including SQL statements, comma- or tab-separated lists, and grid or vertical output, similar to how the `mysql` client displays data. You can redirect this to a file for later use. However, these alternative formats require the use of `mysqldbimport` to restore them. The second utility is `mysqldbimport`. This utility reads the output produced by the `mysqldbexport` utility.

Both utilities allow you to import only the object definitions, only the data, or both. This may sound similar to `mysqldump` and `mysqlpump`, and in many ways that it is true, but these utilities have simplified options (one criticism of the client utilities is the vast array of options available), and since they are written in Python, database professionals can tailor the export and import to their own needs by modifying the code directly.

Listing 7-3 shows an example of running the `mysqlbackup` utility. Notice that the output also resembles the previous client utilities.

---

[6]https://en.wikipedia.org/wiki/DevOps

Listing 7-3. Using MySQL Utilities for Backup

```
$ mysqldbexport --server=root@localhost:3306 plant_monitoring --format=CSV --no-headers
--export=both
# Source on localhost: ... connected.
# Exporting metadata from plant_monitoring
# TABLES in plant_monitoring:
`plant_monitoring`,`plants`,InnoDB,1,`id`,int(11),NO,,PRI,latin1_swedish_
ci,,,,,,,`PRIMARY`,`id`,,
`plant_monitoring`,`plants`,InnoDB,2,`name`,char(50),YES,,,latin1_swedish_
ci,,,,,,,`PRIMARY`,`id`,,
`plant_monitoring`,`plants`,InnoDB,3,`location`,char(30),YES,,,latin1_swedish_
ci,,,,,,,`PRIMARY`,`id`,,
`plant_monitoring`,`plants`,InnoDB,4,`climate`,"enum('inside','outside')",YES,inside,,
latin1_swedish_ci,,,,,,,`PRIMARY`,`id`,,
# FUNCTIONS in plant_monitoring:
`max_samples_today`,SQL,READS_SQL_DATA,YES,DEFINER,root@localhost,in_id int,int(11),"BEGIN
  DECLARE num_samples int;
  SELECT COUNT(*) into num_samples FROM plant_monitoring.readings
  WHERE DATE(event_time) = CURRENT_DATE() AND readings.id = in_id;
  RETURN num_samples;
END","ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_
DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION",utf8,utf8_general_ci,
latin1_swedish_ci
# PROCEDURES in plant_monitoring: (none found)
# VIEWS in plant_monitoring: (none found)
# TRIGGERS in plant_monitoring: (none found)
# EVENTS in plant_monitoring: (none found)
# GRANTS in plant_monitoring: (none found)
#...done.
# Exporting data from plant_monitoring
# Data for table `plant_monitoring`.`plants`:
`id`,`name`,`location`,`climate`
1,Jerusalem Cherry,deck,outside
2,Moses in the Cradle,patio,outside
3,Peace Lilly,porch,inside
4,Thanksgiving Cactus,porch,inside
5,African Violet,porch,inside
#...done.
```

Like the other clients, the MySQL Utilities export and import tools support a number of options for controlling the export and import (although not nearly as many). You can find a more in-depth examination of mysqldbexport and mysqldbimport in the online MySQL Utilities reference manual (http://dev.mysql. com/doc/index-utils-fabric.html).

MySQL Enterprise Backup

If you are looking for a backup solution that permits you to make nonblocking, physical-level backups of your data and you have used the InnoDB storage engine as your primary data storage, you will want to examine the MySQL Enterprise Backup product from Oracle. MySQL Enterprise Backup is part of the MySQL Enterprise Edition product offering. You can find more information about the MySQL Enterprise offerings at http://dev.mysql.com/doc/index-enterprise.html.

While MySQL Enterprise Backup is a fee-based application, you can download a trial version from Oracle's eDelivery system. To do so, go to https://edelivery.oracle.com/, enter **MySQL Enterprise Edition**, and select your platform(s) from the Select Platform drop-down list; then click Continue and follow the prompts. You must have an Oracle web account to access this system. If you do not have such an account, you can create one on the site.

For more information about MySQL Enterprise Backup and other MySQL Enterprise Edition offerings, see http://dev.mysql.com/doc/index-enterprise.html.

Physical File Copy

Finally, the easiest and most basic way to back up MySQL is to simply copy the files. Unfortunately, this requires stopping the server, which may not be ideal. To perform a file copy, stop your server and copy the data directory and any setup files on the server. One common method for this is to use the Unix tar command to create an archive. You can then move this archive to another system and restore the data directory.

The data directory is where MySQL saves all the data in the databases. You can discover its location with the following command:

```
mysql> SHOW VARIABLES LIKE 'datadir';
+---------------+------------------------+
| Variable_name | Value                  |
+---------------+------------------------+
| datadir       | /usr/local/mysql/data/ |
+---------------+------------------------+
1 row in set, 0 warning (0.00 sec)
```

Once the copy is complete, you should move the file to a save location (not on the same disk). To restore the data, you must once again stop the server and then copy the file to the data directory and un-tar it.

■ **Tip**   It is always a good idea to use meaningful file names for your backup images.

The downside of this type of backup is it makes a complete copy of all databases and all data. That may be fine for the general case but may not be the best for restoring only one of several databases. You cannot simply copy the files for each database (they're stored in folders) because if you use InnoDB and have not turned on the file-per-table option, all data is stored in the files named ib*.

Additionally, depending on the size of the data, your server must be offline not only for the time to copy the files, but also for any additional data loads such as cache entries, the use of memory tables for fast lookups, and so on. For this reason, physical copy backup may not be feasible for some installations.

Backup and Recovery with the Binary Log

If your recovery goals are to be able to recover data changed or added since the last backup and your backup solution does not provide incremental or similar up-to-the-minute backup, you run the risk of losing new data. While this sounds like you need to plan your backups for the minimal time you can afford to lose data (which is always a good policy), you can achieve up-to-the-minute recovery by combining planned backups with the binary logs.

Binary logs are special files that contain a copy of all changes made to the databases. In fact, binary logging is the primary mechanism used in replication to capture and transport changes from the master to the slave. You turn on binary logging by using the following option in your configuration file (for example, my.cnf). The option takes a filename prefix that you can use to name the binary logs (this is important, as you will see later in the replication primer).

```
[mysqld]
log-bin=mysql-bin
```

Since the binary log records all changes that are made to the data while the database is running, by restoring the correct backup and playing back the binary log up to the appropriate moment of the failure, you can restore a server to a precise moment in time. This is called *point-in-time recovery* (http://dev.mysql.com/doc/refman/5.7/en/point-in-time-recovery.html). You can replay the binary logs with a client utility named mysqlbinlog.

However, this works only if you keep track of the binary log and position of the last backup. The best way to do this is to flush the logs prior to doing the backup. The resulting new file is the start of the recovery (the first log for new changes).

■ **Note**   There is a newer type of replication that uses global transaction identifiers (GTIDs) that are special markers in the binary log to designate the start and origin of the transaction. Unfortunately, GTIDs are available only in MySQL 5.6 and later. Most versions of MySQL available in the default repositories are MySQL version 5.1 or 5.5, which do not support GTIDs. If you want to use GTIDs, see the online MySQL Reference Manual for more details (http:// dev.mysql.com/doc/refman/5.7/en/replication-gtids.html).

Once you repair the server, you can restore the latest backup image and apply the binary log using the last binary log name and position as the starting point. The following describes one procedure you can use to perform point-in-time recovery using the backup system:

1. Return your server to an operational state after the event.

2. Find the latest backup for the databases you need to restore.

3. Restore the latest backup image.

4. Apply the binary log with the mysqlbinlog utility using the starting position (or starting date/time) from the last backup.

After the binary logs are applied, your server has been recovered, and you can return it to service. Of course, this is a good time to make another backup![7]

■ **Tip**   For easier point-in-time recovery, always flush the logs prior to a backup.

---

[7]No, really. You'd be surprised how handy such a backup could be if the repair fails soon after you start using the server—an event that is all too common in the IT world.

MySQL Replication Primer

One of the nicest things about using an external drive to save your MySQL data is that at any point you can shut down your server, disconnect the drive, plug it in to another system, and copy the data. That may sound great if your Raspberry Pi database server is in a location that makes it easy to get to (physically) and if there are periods when it is OK to shut down the server.

However, this may not be the case for some IOT networks. One of the benefits of using a low-cost computer board like a Raspberry Pi for a database server is that the server can reside in close proximity to the data collector nodes. If part of the IOT network is in an isolated area, you can collect and store data by putting the Raspberry Pi in the same location. But this may mean trudging out to a barn or pond or walking several football field lengths into the bowels of a factory to get to the hardware if there is no network to connect to your database server.

But if your Raspberry Pi is connected to a network, you can use an advanced feature of MySQL called *replication* to make a live, up-to-the-minute copy of your data. Not only does this mean you can have a backup, but it means you can query the server that maintains the copy and therefore unburden your Raspberry Pi of complex or long-running queries. It also means you can have a hot standby should the first server (master) fail. That is, you can switch to the copy (slave) and keep your application running. The Raspberry Pi is a cool small-footprint computer, but a data warehouse it is not.

What Is Replication, and How Does It Work?

MySQL replication is an easy-to-use feature and yet a complex and major component of the MySQL server. This section presents a bird's-eye view of replication for the purpose of explaining how it works and how to set up a simple replication topology. For more information about replication and its many features and commands, see the online MySQL reference manual (http://dev.mysql.com/doc/refman/5.7/en/replication.html).

Replication requires two or more servers. One server must be designated as the origin or master. The master role means all data changes (writes) to the data are sent to the master and only the master. All other servers in the topology maintain a copy of the master data and are by design and requirement read-only servers. Thus, when your sensors send data for storage, they send it to the master. Applications you write to use the sensor data can read it from the slaves.

The copy mechanism works using a technology called the *binary log* that stores the changes in a special format, thereby keeping a record of all the changes. These changes are then shipped to the slaves and reexecuted there. Thus, once the slave reexecutes the changes (called *events*), the slave has an exact copy of the data.

The master maintains a binary log of the changes, and the slave maintains a copy of that binary log called the *relay log*. When a slave requests data changes from the master, it reads the events from the master and writes them to its relay log; then another thread in the slave executes those events from the relay log. As you can imagine, there is a slight delay from the time a change is made on the master to the time it is made on the slave. Fortunately, this delay is almost unnoticeable except in topologies with high traffic (lots of changes). For your purposes, it is likely when you read the data from the slave, it is up-to-date. You can check the slave's progress using the command SHOW SLAVE STATUS; among many other things, it shows you how far behind the master the slave has become. You see this command in action in a later section.

Now that you have a little knowledge of replication and how it works, let's see how to set it up. The next section discusses how to set up replication with the Raspberry Pi as the master and a desktop computer as the slave.

How to Set Up Replication

This section demonstrates how to set up replication from a Raspberry Pi (master) to a desktop computer (slave). The steps include preparing the master by enabling binary logging and creating a user account for reading the binary log, preparing the slave by connecting it to the master, and starting the slave processes. The section concludes with a test of the replication system.

Preparing the Master

Replication requires the master to have binary logging enabled. It is not turned on by default, so you must edit the configuration file and turn it on. Edit the configuration file with sudo vi /etc/mysql/my.cnf, and turn on binary logging by uncommenting and changing the following lines:

```
server-id              = 1
log_bin                = /media/HDD/mysql/mysql-bin.log
```

The first line sets the server ID of the master. In basic replication (what you have for version 5.5), each server must have a unique server ID. In this case, you assign 1 to the master; the slave will have some other value, such as 2. Imaginative, yes?

The next line sets the location and name of the binary log file. You save it to your external drive because, like the data itself, the binary log can grow over time. Fortunately, MySQL is designed to keep the file to a reasonable size and has commands that allow you to truncate it and start a new file (a process called *rotating*). See the online reference manual (http://dev.mysql.com/doc/refman/5.5/en/slave-logs-relaylog.html) for more information about managing binary log files.

Once the edits are saved, you can restart the MySQL server with the following command:

```
pi@raspberrypi /etc $ sudo /etc/init.d/mysql restart
[ ok ] Stopping MySQL database server: mysqld.
[ ok ] Starting MySQL database server: mysqld . . ..
[info] Checking for tables which need an upgrade, are corrupt or were
not closed cleanly..
```

To test the change, issue the following command in a MySQL console. You should see that the new variable has been set to ON.

```
mysql> show variables like 'log_bin';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_bin       | ON    |
+---------------+-------+
1 row in set (0.01 sec)
```

After binary logging is turned on, you must create a user to be used by the slave to connect to the master and read the binary log. There is a special privilege for this named REPLICATION SLAVE. The following shows the correct GRANT statement to create the user and add the privilege. Remember the username and password you use here—you need it for the slave.

```
mysql> GRANT REPLICATION SLAVE ON *.* TO 'rpl'@'%' IDENTIFIED BY 'secret';
Query OK, 0 rows affected (0.01 sec)
```

But one more piece of information is needed for the slave. The slave needs to know the name of the binary log to read and what position in the file to start reading events. You can determine this with the SHOW MASTER STATUS command.

```
mysql> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |      245 |              |                  |
+------------------+----------+--------------+------------------+
1 row in set (0.00 sec)

mysql>
```

Now that you have the master's binary log file name and position as well as the replication user and password, you can visit your slave and connect it to the master. You also need to know the hostname or IP address of the Raspberry Pi as well as the port on which MySQL is running. By default, the port is 3306; but if you changed that, you should note the new value. Jot down all that information in Table 7-3.

Table 7-3. *Information Needed from the Master for Replication*

Item from Master	Value
IP address or hostname	
Port	
Binary log file	
Binary log file position	
Replication user ID	
Replication user password	

Preparing the Slave

The MySQL server you want to use as a slave should be the same version as the server on the Raspberry Pi, or at least a server that is compatible. The online reference manual specifies which MySQL versions work well together. Fortunately, the list of versions with issues is very short. In this section, you should have a server installed on your desktop or server computer and ensure that it is configured correctly.

The steps needed to connect a slave to a master include issuing a CHANGE MASTER TO command to connect to the master and a START SLAVE command to initiate the slave role on the server. Yes, it is that easy! Recall that you need the information from the master to complete these commands. The following commands show a slave being connected to a master running on a Raspberry Pi. Let's begin with the CHANGE MASTER TO command.

Listing 7-4. Using the CHANGE MASTER TO Command

```
Chucks-iMac:~ cbell$ mysql -uroot -psecret -h 127.0.0.1 --port=13003
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.5.21 Source distribution
```

```
Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CHANGE MASTER TO MASTER_HOST='10.0.1.17', MASTER_PORT=3306, MASTER_LOG_FILE=
'mysql-bin.000001', MASTER_LOG_POS=245, MASTER_USER='rpl', MASTER_PASSWORD='secret';
Query OK, 0 rows affected (0.22 sec)
```

This example uses the IP address of the Raspberry Pi, the port number (3306 is the default), the log file and position from the SHOW MASTER STATUS command, and the username and password for the replication user. If you typed the command correctly, it should return without errors. If there are errors or warnings, use the SHOW WARNINGS command to read the warnings and correct any problems.

The next step is to start the slave processes. This command is simply START SLAVE. It normally does not report any errors; you must use SHOW SLAVE STATUS to see them. Listing 7-5 shows both of these commands in action.

■ **Tip** For wide results, use the \G option to see the columns as rows (called *vertical format*).

Listing 7-5. Starting the Slave

```
mysql> start slave;
Query OK, 0 rows affected (0.00 sec)
mysql> show slave status \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.0.1.17
                  Master_User: rpl
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 107
               Relay_Log_File: clone-relay-bin.000003
                Relay_Log_Pos: 4
        Relay_Master_Log_File: mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 107
              Relay_Log_Space: 555
```

```
                  Until_Condition: None
                   Until_Log_File:
                    Until_Log_Pos: 0
               Master_SSL_Allowed: No
               Master_SSL_CA_File:
               Master_SSL_CA_Path:
                  Master_SSL_Cert:
                Master_SSL_Cipher:
                   Master_SSL_Key:
            Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                    Last_IO_Errno: 0
                    Last_IO_Error:
                   Last_SQL_Errno: 0
                   Last_SQL_Error:
     Replicate_Ignore_Server_Ids:
                 Master_Server_Id: 1
1 row in set (0.00 sec)

mysql>
```

Take a moment to slog through all these rows. There are several key fields you need to pay attention to. These include anything with error in the name, and the state columns. For example, the first row (Slave_IO_State) shows the textual message indicating the state of the slave's I/O thread. The I/O thread is responsible for reading events from the master's binary log. There is also a SQL thread that is responsible for reading events from the relay log and executing them.

For this example, you just need to ensure that both threads are running (YES) and there are no errors. For detailed explanations of all the fields in the SHOW SLAVE STATUS command, see the online MySQL reference manual (http://dev.mysqlcom/doc) in the section "SQL Statements for Controlling Slave Servers."

Now that the slave is connected and running, let's check for that testme database on it.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.00 sec)

mysql>
```

Wait! Where did it go? Wasn't this example supposed to replicate everything? Well, yes and no. It is true that your slave is connected to the master and will replicate anything that changes on the master from this point on. Recall that you used the SHOW MASTER STATUS command to get the binary log file and position. These values are the coordinates for the location of the next event, not any previous events. Aha—you set up replication *after* the testme database was created.

How do you fix this? That depends. If you really wanted the testme database replicated, you would have to stop replication, fix the master, and then reconnect the slave. I won't go into these steps, but I list them here as an outline for you to experiment on your own:

1. Stop the slave.

2. Go to the master and drop the database.

3. Get the new SHOW MASTER STATUS data.

4. Reconnect the slave.

5. Start the slave.

Got that? Good. If not, it is a good exercise to go back and try these steps on your own.

Once you get the master cleaned and replication restarted, go ahead and try to create a database on the master and observe the result on the slave. Listing 7-6 shows the commands. Note that I used a different database name in case you elected to not try the previous challenge.

Listing 7-6. Testing Replicaiton of New Database on the Slave

```
pi@raspberrypi /etc $ mysql -uroot -psecret
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 38
Server version: 5.5.28-1-log (Debian)

Copyright (c) 2000, 2012, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database testme_again;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| testme             |
| testme_again       |
+--------------------+
4 rows in set (0.01 sec)

mysql>
```

Returning to the slave, check to see what databases are listed there, as shown in Listing 7-7.

Listing 7-7. Verifying New Database Is on the Slave

```
Chucks-iMac:mysql-5613 cbell$ mysql -uroot -psecret -h 127.0.0.1 --port=13003
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 5.5.21 Source distribution

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| testme_again       |
+--------------------+
4 rows in set (0.00 sec)

mysql>
```

Success! Now your Raspberry Pi database server is being backed up by your desktop computer.

IS THERE A BETTER WAY?

If you are wondering whether there is an easier way to get replication working without fiddling around with commands on the slaves, then I have good news. There is a better way! See the `mysqlreplicate` utility in MySQL Utilities. This utility allows you to set up replication with a single command. For more information about `mysqlreplicate`, see the online MySQL Utilities documentation at http://dev.mysql.com/doc/index-gui.html.

Fault Tolerance in IOT Nodes

Achieving fault tolerance with MySQL is not overly difficult and as you have seen can be accomplished using MySQL replication for redundancy, recovery, scalability, and ultimately high availability for your database component. However, achieving fault tolerance on a typical microcontroller-based node that writes data to the database can be a bit more difficult.

This is because you will have to write all the code to detect faults and implement redundant mechanisms to recover from the fault. Sometimes this can result in the code being far larger than what a typical microcontroller has room to store. Thus, you should consider fault tolerance on microcontrollers to be costly in terms of memory. Fortunately, there are microcontrollers with more memory that you can use.

You may be wondering why you would implement fault tolerance at this level. Recall the goal is to preserve the data, in other words, to collect and store it in a timely manner. Some IOT solutions can afford to lose or miss data samples over a short period of time, but other IOT solutions—particularly those dealing with time-sensitive or health data—may rely on the data for accuracy in making predictions (e.g., diagnosis) or recommending courses of action. For these IOT solutions, you will want to implement fault tolerance throughout the solution, from the sensors to the data collectors to the database server and even the application server.

You will see a full implementation of a fault-tolerant data collector in the next chapter. As you will see, the complexity I warned about is there, and we will have to make some concessions with the hardware to get it to work.

Summary

High availability isn't just for large, monolithic applications written for large organizations. High availability is also not difficult to achieve at more modest levels of reliability. In fact, there are many tools you can use to make your IOT solutions more reliable.

In this chapter, you learned what high availability is and how high availability concepts can be realized. You also learned key high availability concepts, tools, and techniques for MySQL including backup and recovery and replication. You also discovered how to implement fault tolerance for collecting data on microcontrollers.

In the next chapter, you will see a demonstration of advanced techniques for building IOT solutions with MySQL.

CHAPTER 8

■ ■ ■ ■

Demonstration of High Availability Techniques

Understanding what high availability is and some of the key goals for achieving higher levels of reliability will help you design your IOT solutions for greater reliability. As you learned in the previous chapter, it takes a bit of work to achieve these goals, but the payoff for a modest amount of work is well worth it should your solution expand beyond a set of discrete nodes.

However, knowing the concepts is only part of the solution. You also need to see examples of how some of these techniques are manifested. For example, how do you detect and recover from a data collector node failure, and how do you set up MySQL replication to switch to a new master should the master fail? Furthermore, how does the application know this has happened?

We will explore these topics and more using example projects that illustrate the techniques in code. Some of the examples are written in Python since these are more likely to be implemented on low-cost computer boards or even personal or server computers. Other examples are written for the Arduino but could also be adapted to other microcontroller boards.

■ **Tip** The examples in this chapter present techniques rather than complete solutions. They are more of a recipe for success than a how-to demonstration. Even so, the examples in this section are advanced techniques that you can employ in your own solutions.

Before we jump into the code examples, let's explore the primary tool for making your database server more reliable: MySQL replication.

MySQL Replication Techniques

Recall MySQL replication is a mechanism that permits you to designate one server as the master and one or more servers as slaves whereby the slaves receive copies of all changes made to the master. This permits you to replicate the data from one server to many others. In the previous chapter, you saw how to set up MySQL replication for hot standby or backup and manual recovery should the master fail.

While MySQL replication is easy to set up[1] and managing a small replication topology is normally not a burden, requiring very little intervention, managing large replication topologies and solving problems that can occur—both intentional and unintentional—can be more of a challenge.

---

[1]Made easier with MySQL Utilities, as you will see.

■ **Note** The replication examples in this chapter require MySQL server version 5.6.5 or higher.

Experience with using MySQL replication will help overcome many of the normal needs you may have along the way. Indeed, a quick read through the replication section in the online MySQL reference manual will help considerably. However, there are a few things that can occur that are less obvious in the documentation (but are described, just not called out so much). I cover these topics using examples in this section starting with failover and switchover.

You will also examine a number of advanced MySQL replication techniques, best practices, and even some examples of how to leverage MySQL replication in your IOT solutions. Let's begin with a collection of key techniques to get the most out of MySQL replication; first up is a key database concept called *transaction processing*.

Transaction Processing

A transaction is a set of operations on a database that you want to make atomic. That is, if one fails, you want all the changes since the start of the transaction to be reversed (reverted). The classic example is moving money from one bank account to another. This requires deducting the amount from the first account and adding it to the second—an act that requires two steps. If either should fail, neither of those changes should occur; otherwise, you've got money unallocated or allocated without a transfer!

We start a transaction with the special START TRANSACTION or BEGIN statement. Should all statements succeed, we issue a COMMIT statement to complete the action and instruct the database server to write the changes permanently. Should we want to revert the changes, we can issue a ROLLBACK statement to undo all the changes.

Transactions are supported in MySQL when using the InnoDB storage engine. This engine is the default storage engine in MySQL, so transactions are enabled by default.

Listing 8-1 shows an example of this in action. We start with a test table containing only two columns—an auto-increment column and an integer column. Observe how transactions can be used to preserve data.

Listing 8-1. Demonstration of Transaction Processing in MySQL

```
mysql> USE test;
Database changed
mysql> CREATE TABLE test_trans(id INT AUTO_INCREMENT PRIMARY KEY, value INT);
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO test_trans VALUES (NULL,1), (NULL,2), (NULL,3);
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO test_trans VALUE (NULL,4);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM test_trans;
+----+-------+
| id | value |
+----+-------+
|  1 |     1 |
|  2 |     2 |
|  3 |     3 |
|  4 |     4 |
+----+-------+
4 rows in set (0.00 sec)

mysql> DELETE FROM test_trans;
Query OK, 4 rows affected (0.00 sec)

mysql> SELECT * FROM test_trans;
Empty set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM test_trans;
+----+-------+
| id | value |
+----+-------+
|  1 |     1 |
|  2 |     2 |
|  3 |     3 |
+----+-------+
3 rows in set (0.01 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO test_trans VALUES (NULL,4), (NULL,5);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> DELETE FROM test_trans WHERE id <= 2;
Query OK, 2 rows affected (0.01 sec)

mysql> SELECT * FROM test_trans;
+----+-------+
| id | value |
+----+-------+
|  3 |     3 |
|  5 |     4 |
|  6 |     5 |
+----+-------+
3 rows in set (0.00 sec)
```

```
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM test_trans;
+----+-------+
| id | value |
+----+-------+
|  3 |     3 |
|  5 |     4 |
|  6 |     5 |
+----+-------+
3 rows in set (0.00 sec)
```

Notice we begin by inserting several rows without transactions. In this case, the MySQL server is set up to automatically commit these statements. This is called *auto commit* and can be controlled with the autocommit option. Next, we start a transaction executing a number of statements and then roll back the operation and check the results. Finally, you can see an example of how to commit.

One thing that may not be apparent with transaction processing is how the changes affect other connections. For example, in the previous demonstration, what would happen if another connection were to query the data before the transaction is committed? The connection would see the original, unaltered data because it hasn't been committed yet. Interestingly, the current connection would see the changes before the commit because the changes are local to that session. You can see that in the SELECT in the middle of the transaction.

Transactions are an excellent advanced technique to use in your solutions, especially once you develop complex, multistep statements for your data. Also, transactions are a key concept used in more advanced replication features such as global transaction identifiers.

Advanced Replication with Global Transaction Identifiers

Recall from Chapter 7, you discovered how to set up replication using a mechanism that requires identifying the binary log file and position from the master and then specifying that information on the slave with the CHANGE MASTER TO statement. This is referred to as *binary log file* and *position replication*. However, there is a newer, better way that improves performance and makes possible automatic failover and easier switchover. The feature required is called *replication* with global transaction identifiers (GTIDs) or GTID-based replication.

GTIDs enable servers to assign a unique identifier to each set or group of events, thereby making it possible to know which events have been applied on each slave. That is, when GTIDs are enabled, the replication component of the MySQL server will inject a special event into the binary log that contains a globally unique identifier that is unique to each server and a sequence number at the start of a transaction. In this way, MySQL replication can detect what events are associated with each transaction and even from which server transactions were sent. Best of all, the replication protocol can now determine which transactions are missing on a slave so that when a slave connects, only those transactions that are not already applied to the slave get sent across.

Now let's see how to set up a master and several slaves using GTIDs. If your servers already have data on them, you should synchronize the data on all servers before turning on GTIDs. However, before we turn on GTIDs, you should create a user on the master that the slaves can use to connect. This user must have

the REPLICATION SLAVE privilege for all databases. It is also good practice to assign the user a password. The following statements show how to set up the replication user account and grant the appropriate privileges:

```
CREATE USER 'rpl'@'localhost' IDENTIFIED BY 'rpl';
GRANT REPLICATION SLAVE ON *.* TO 'rpl'@'localhost';
```

Now we are ready to set up the servers to use GTIDs. To do so, you must specify the following options for the master. You can place these in the my.cnf file or specify them as startup commands.

```
[mysqld]
gtid-mode=ON
enforce-gtid-consistency
```

Here, we turn on GTIDs and then add an additional option to enforce consistency across servers. You will want to use this option if you plan to switch the master role or plan to set up failover.

On the slaves, we also need to specify the following options in the my.cnf file or specified as startup commands. These options are required for all servers in the topology. In addition, the slaves can be configured to use a table to store the master information. You can see these options in the following sample configuration file for each slave:

```
[mysqld]
gtid-mode=ON
enforce-gtid-consistency
master-info-repository=TABLE
report-host=slave1
report-port=13002
```

■ **Note** If you ever plan to use your master as a slave, you should use the options specified for the slave.

Once all the servers have been restarted and you have confirmed there are no errors, you can begin configuring replication. Recall from Chapter 7, we issue the CHANGE MASTER TO and START SLAVE statements on each slave. The following presents an example of the statements:

```
CHANGE MASTER TO MASTER_HOST='master', MASTER_PORT=13001, MASTER_USER='rpl', MASTER_
PASSWORD='secret', MASTER_AUTO_POSITION = 1;
START SLAVE;
```

Notice here we no longer need to specify the master binary log file and position. Instead, we simply instruct the slave to use the automatic position feature provided by GTIDs. You can check the status of replication on the slave using SHOW SLAVE STATUS. You should not see any errors. For more information about GTIDs, see the online MySQL reference manual at http://dev.mysql.com/doc/refman/5.7/en/replication-gtids.html.

■ **Tip** Use the \G option to see a vertical list of values instead of a tabular output. It makes wide rows easier to read and, in this case, the output readable by mere mortals.

Now that you understand and can use GTIDs, you can start using some advanced replication concepts such as failover.

TESTING REPLICATION WITH MYSQL UTILITIES

In the following sections, I present several concepts and examples that use replication. To run the examples, particularly the scaling and failover examples, you will need to set up a replication topology of one master and two or three slaves. The following are steps you can take to set up a test replication topology on your own system. Documentation for each of these steps and examples is included in the online MySQL Utilities documentation (http://dev.mysql.com/doc/mysql-utilities/1.6/en/).

1. Clone a running (installed) MySQL server with `mysqlreplicate`. Make at least three or four clones assigning the correct port, server ID, database director, and so on (change all occurrences of `13001` in the command).

   ```
   mysqlserverclone --server=root:secret@localhost:3306
   --new-data=/tmp/13001
   --new-port=13001 --new-id=1 --root-password=root --del
   --mysqld="--log_bin=mysql-bin
   --gtid-mode=on --enforce-gtid-consistency
   --master-info-repository=table
   --report-host=localhost --report-port=13001"
   ```

2. Set up the replication user on each server. Use the `mysql` client to connect to each server in turn and run the following SQL statements:

   ```
   SET @@sql_log_bin=0;
   CREATE USER 'rpl'@'localhost' IDENTIFIED BY 'rpl';
   GRANT REPLICATION SLAVE ON *.* TO 'rpl'@'localhost';
   SET @@sql_log_bin=1;
   ```

3. Once all servers are cloned, set up replication with `mysqlreplication`. Here I use the server on port 13001 as the master. Run this command once for each slave (change the slave port accordingly).

   ```
   mysqlreplicate --master=root:root@localhost:13001
   --slave=root:root@localhost:13002
   --rpl-user=rpl:rpl
   ```

4. To see the topology, use `mysqlrplshow` using the server and the discover slaves login option.

   ```
   mysqlrplshow --master=root:root@localhost:13001 --disco=root:root
   ```

Wasn't that a lot easier than doing it manually? I have included with the source code for the book a file named `setup_topology.txt` that demonstrates commands you can use to create the topology with MySQL Utilities. Just change the options to match your system.

Switchover

Sometimes it is necessary to switch the role of master from the original master to one of the slaves. You may need to do this to perform maintenance on the master, recover from data loss, and perform other events that require stopping replication for a period of time. Rather than force your solution to stop altogether, you can simply switch the master role temporarily and then switch it back when the master is back online.

What follows is a demonstration of performing switchover using MySQL Utilities. Recall MySQL Utilities consists of a set of Python scripts and libraries designed to help automate server maintenance. Many of the utilities are designed to assist in setting up and managing replication. Listing 8-2 shows an execution of the mysqlrpladmin utility to perform switchover from the master to one of the slaves. The output shows you all the steps taken. Had you done this manually, you would have had to do each yourself on each slave and then the master.

Listing 8-2. Switchover Using mysqlrpladmin

```
$ mysqlrpladmin switchover --demote-master --master=root:secret@localhost:13001 --new-
master=root:secret@localhost:13002 --discover-slaves=root:secret
# Discovering slaves for master at localhost:13001
# Discovering slave at localhost:13002
# Found slave: localhost:13002
# Discovering slave at localhost:13003
# Found slave: localhost:13003
# Discovering slave at localhost:13004
# Found slave: localhost:13004
# Checking privileges.
# Performing switchover from master at localhost:13001 to slave at localhost:13002.
# Checking candidate slave prerequisites.
# Checking slaves configuration to master.
# Waiting for slaves to catch up to old master.
# Stopping slaves.
# Performing STOP on all slaves.
# Demoting old master to be a slave to the new master.
# Switching slaves to new master.
# Starting all slaves.
# Performing START on all slaves.
# Checking slaves for errors.
# Switchover complete.
#
# Replication Topology Health:
+-----------+-------+---------+--------+------------+---------+
| host      | port  | role    | state  | gtid_mode  | health  |
+-----------+-------+---------+--------+------------+---------+
| localhost | 13002 | MASTER  | UP     | ON         | OK      |
| localhost | 13001 | SLAVE   | UP     | ON         | OK      |
| localhost | 13003 | SLAVE   | UP     | ON         | OK      |
| localhost | 13004 | SLAVE   | UP     | ON         | OK      |
+-----------+-------+---------+--------+------------+---------+
# ...done.
```

To perform switchover, I specify the original master (recall switchover requires a healthy, running master), the new master, and the login and password for each slave (called *discovery*). The options I specified to retain the master as a slave are what are interesting here. The demote-master option returns the master as
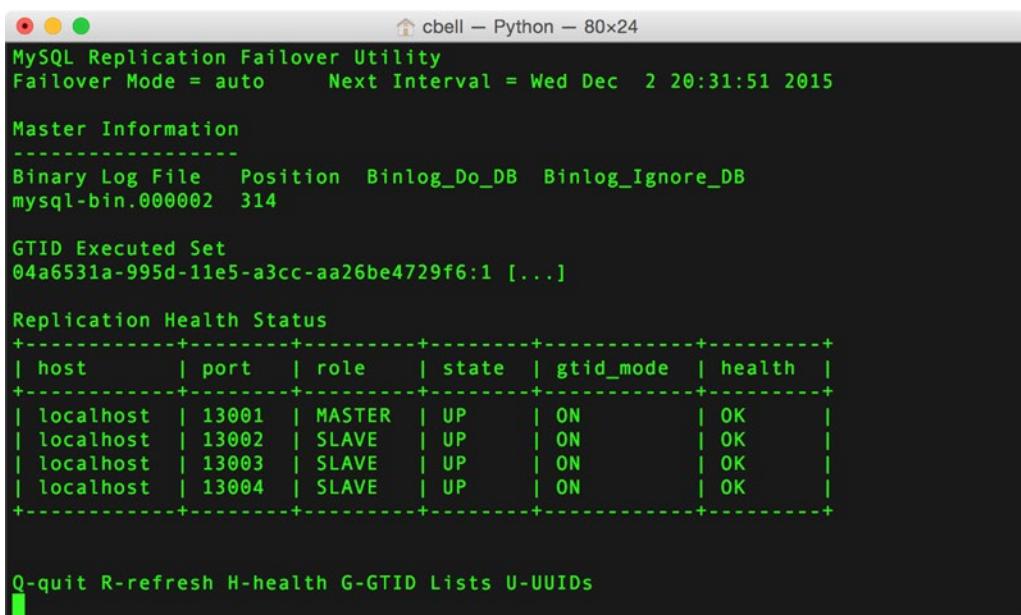
a slave to the topology after the new master takes over. It should be noted here that this is not possible unless you set up the master like a slave (with report-host, report-port, and master-into-repository). You can see the results at the end of the listing where the utility prints out a health report for the new topology. Here you see the original master (on port 13001) is now a slave.

Failover

Failover is a situation where the master has failed in some manner and you need to automatically reconfigure the topology to recover the master role. Prior to the use of GTIDs, automatic failover was problematic at best and unreliable for all but the simplest of cases.

To perform failover with GTIDs, we choose the best slave (the one with the least missing events and the hardware that matches the master best) and make it a slave of every other slave (also called *slave promotion*). We call this slave the *candidate slave*. The GTID mechanism will ensure only those events that have not been executed on the candidate slave are applied. In this way, the candidate slave becomes the most up-to-date and therefore a replacement for the master.

Once again, you can use MySQL Utilities to watch the master and automatically promote one of the slaves to the new master. The mysqlfailover command-line tool performs automatic failover by executing the previous sequence of events and takes care of redirecting the remaining slaves to the new master. Figures 8-1 through 8-3 show a series of screenshots taken of the utility. The first shows the utility in monitoring mode.

Figure 8-1. Automatic failover with mysqlfailover (before failover)

Notice the utility displays a number of statistics including the current health report of all the servers in the topology. The utility works by checking the state of the master at a specified interval. In this example, I used an interval of 30 seconds. To simulate the master failing, I killed the master process. Figure 8-2 shows the utility detecting the failure and executing automatic failover.
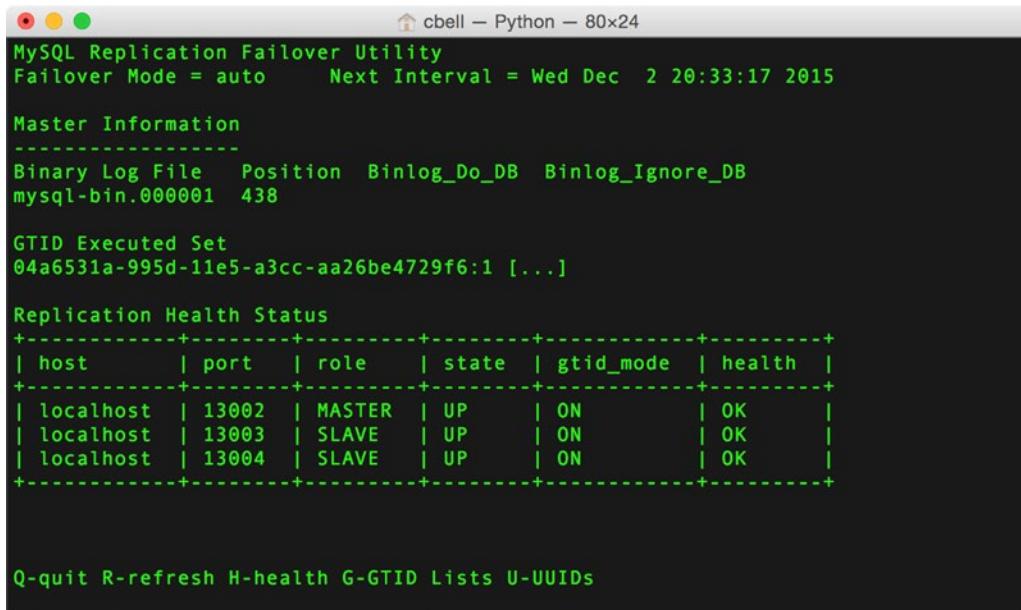
Figure 8-2. *Automatic failover with mysqlfailover (fault detected)*

When the failover is complete, the utility returns to its monitoring mode, as shown in Figure 8-3. Automatic failover really is that easy! And, yes, you can run the utility on a low-cost computer provided Python 2.6 or later and Connector/Python is installed.

Figure 8-3. *Automatic failover with mysqlfailover (after failover)*

Note in the previous screenshots that the utility is running as a console. If you need to run it as a background process, there is a companion utility that permits you to run it as a process or daemon (for non-Windows platforms only). To learn more about `mysqlfailover` and its companion daemon process including all the available commands and more examples of use, see the online manual for MySQL Utilities at http://dev.mysql.com/doc/mysql-utilities/1.6/en/utils-task-autofailover.html.

Replication and Database Maintenance Tips

As mentioned, MySQL replication is relatively easy to set up and maintain for smaller topologies. For IOT solutions, you are not likely to encounter replication topologies with many servers (but you may). Recall MySQL replication can require a bit of work should things go wrong. In this section, I focus on some common things that can go wrong and present methods to prevent and recover from errors.

Avoiding Errant Transactions

Perhaps the most common mistake new administrators make with MySQL replication is introducing errant transactions. Errant transactions are those statements (or transactions) that apply to only one server and either are invalid when executed on another server or cause replication errors when executed. Errant transactions are most often due to running one or more SQL statements on the master that should not be replicated. With GTIDs, errant transactions can occur when the same is performed on a slave and that slave later becomes the master or leaves and rejoins the topology.

For example, if you execute any SQL command on the master to create, update, or delete anything that does not exist on the slaves, you could encounter errors severe enough to stop replicaiton on the slave. Recall that once replication is started, every SQL statement is written to the binary log and then transmitted to the slaves. Thus, we must be careful when executing statements on the master when binary logging is enabled.

Fortunately, there is a way to omit statements from being recorded to the binary log. There is a special variable named `sql_log_bin` that you can turn off for the current session (the logged-in user) that does not affect any other connection. Simply turn the binary log off, execute your statements, and then turn it on again. The following code shows how you can use this technique to add a replication user account to a server that already has the binary log enabled:

```
SET @@sql_log_bin=0;
CREATE USER 'rpl'@'localhost' IDENTIFIED BY 'rpl';
GRANT REPLICATION SLAVE ON *.* TO 'rpl'@'localhost';
SET @@sql_log_bin=1;
```

If you use this technique when performing maintenance on your replication servers, you can avoid errant transactions and thus avoid a lot unwarranted trouble resolving slave errors.

Resolving Slave Errors

When you encounter errors on the slave, you will see the error in the SHOW SLAVE STATUS output. If the error is caused by an errant transaction, you must instruct the master to skip those transactions, which is easy for binary log file and position-based replication but can be a little tricky for GTID-enabled servers. Recall an errant transaction is normally some event in the binary log of the master that does not apply or produces an error when run on the slave.

For binary log file and position-based replication, you can instruct the slave to skip a number of events if they do not apply to the slave. But first, you should perform the following steps for diagnosing the problem. At this point, the slave has stopped with an error.

1. Determine the error and look up the error in the reference manual. Sometimes the error is well known or has a predictable solution.

2. Determine the event or events causing a problem. A look at the output of SHOW SLAVE STATUS will tell you the binary log file and position of the event. You can use that on the master to see the event with the SHOW BINLOG EVENTS statement. If the event cannot be run (wrong tables, and so on), you must skip the event. If you are not sure, start the slave again and check for errors.

3. If you must skip the events, issue the following statements to skip and start the slave. The following statement skips the next two events:

   ```
   mysql> SET GLOBAL sql_slave_skip_counter = 2
   mysql> START SLAVE;
   ```

4. If this does not solve the problem, you may need to skip more events. However, if you cannot find a restarting point, you may need to do more intensive diagnosis or restore the slave from the latest backup and restart replication.[2]

For GTID-enabled replication, things are not so simple. Here, we need to identify the errant transactions on the slaves. You can see these in the SHOW BINLOG EVENTS output on the slave. Once you have the GTID, you can use the following statements to create an empty transaction in the binary log on the master. This will effectively tell the master that the event has already occurred even though it was not executed on the master or any slave to which the event is transmitted.

Errant transactions are more common in GTID replication because the slaves are all logging changes in their own binary logs. Plus, when the slave connects to a master, it exchanges its lists of GTIDs with the master. Should the lists disagree, there will be errant transactions. You can determine whether there are errant transactions by issuing the following statements. We begin by identifying the UUID on the master.

```
mysql> SHOW VARIABLES LIKE "%uuid%";
+---------------+--------------------------------------+
| Variable_name | Value                                |
+---------------+--------------------------------------+
| server_uuid   | 0f5ae7a2-9968-11e5-8990-08fdde5e3c13 |
+---------------+--------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

Then, on the slave, run SHOW SLAVE STATUS and examine the result for executed_gtid_set, as shown here:

```
Executed_Gtid_Set: 0f5ae7a2-9968-11e5-8990-08fdde5e3c13:1,
130a0dec-9968-11e5-9766-aa87a4a44672:1
```

---

[2]Sometimes this is the easiest and fastest way to bring a slave back online after encountering fatal errors or data corruption. Thus, always take regular backups.

Errant transactions on the slave are those GTIDs listed that do not start with the UUID from the master. From this list, we see there is one GTID, 130a0dec-9968-11e5-9766-aa87a4a44672:1, which is errant (not on the master and not originating from the master). To skip this, we need to trick the master into ignoring the transaction. Let's look at the statements and then discuss their use. Listing 8-3 shows the process to create an empty transaction to ignore (skip) an errant transaction.

Listing 8-3. Skipping Errant Transactions with GTIDs

```
mysql> SET GTID_NEXT='dd365ccc-9898-11e5-872d-172ea4aa6911:1';
Query OK, 0 rows affected (0.00 sec)
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
mysql> SET GTID_NEXT='AUTOMATIC';
Query OK, 0 rows affected (0.00 sec)
mysql> FLUSH LOGS;
Query OK, 0 rows affected (0.08 sec)
mysql> SHOW BINARY LOGS;
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mysql-bin.000001 |      1091 |
| mysql-bin.000002 |       314 |
+------------------+-----------+
2 rows in set (0.00 sec)
mysql> PURGE BINARY LOGS TO 'mysql-bin.000002';
Query OK, 0 rows affected (0.04 sec)
```

Wow! That's a complex process. We begin with setting the next GTID on the master and then starting and committing a transaction (an empty one). We then return the GTID numbering to automatic and flush the logs. We do this so that we can purge the old logs. You can see in the output of the SHOW BINARY LOGS that we can purge the old binary log file. Once this process is complete, we can restart the slave.

For more information about this process, see the online MySQL reference manual at http://dev.mysql.com/doc/refman/5.7/en/replication-gtids-failover.html#replication-gtids-failover-empty.

Starting Replication with Existing Data

The discussion of replication in Chapter 7 assumed you were starting replication from a clean slate. However, if you already have data on the master, you will need to make a backup of the data on the master and restore it on the slave before starting the binary log on the master.

However, what if you already have a master and slave running and have been for some time but you want to add another slave? You could use the backup you made originally when you set up the slave (if you did so), but the new slave will have to read all the changes that have occurred since the backup.

You could also take a backup of the master, but you would have to suspend writing to the binary log to do so.[3] A better method is to temporarily stop the existing slave and wait until it has read and applied the events in its relay log (use the SHOW SLAVE STATUS statement to see the current state). The columns you need to view are highlighted here using an excerpt from an actual replication slave:

```
Slave_IO_State: Waiting for master to send event
...
Slave_SQL_Running_State: Slave has read all relay log; waiting for more updates
```

Here you can see the slave has read all events from the master (slave_io_state) and the slave has processed all the events in its relay log (slave_sql_running_state).

If you use binary log file and position-based replication, you will need to record the current master binary log file and position as shown in the SHOW SLAVE STATUS output. If you use GTIDs, you do not need this information.

Once the slave is stopped and all relay log events have been executed, you can make the backup, restart the original slave, then restore the data on the new slave, and finally start replication on the new slave.

Example: Scaling Applications

Scaling is about performance. Read scaling is where you use additional servers to direct reads (SELECT statements) so that no one server is burdened with processing a lot of read requests. There is also shard scaling where you divide (partition) the data among several servers ideally so that they all have the same amount of data. To access the shard, you must use an algorithm (commonly a simple hash function) or a range of values dedicated to each shard. As you can imagine, it can be a challenge to route the query to the correct shard. Fortunately, most IOT solutions will not require the use of sharding for improving performance. Rather, it is more likely some IOT solutions can benefit from read scaling.

Read scaling with MySQL uses a replication topology with all the writes (CREATE, INSERT, UPDATE, DELETE, and so on) sent to the master and all reads sent to one of the slaves. The number of slaves you will need will be determined largely by how often (or much) your application reads data.

While setting up read scaling is no more difficult than setting up replication (you don't need to do anything special), writing your application to scale the reads is the challenge. More specifically, knowing which server to send a client (application) can be a simple round-robin or queue mechanism, or you could implement sophisticated load balancing to direct the reads to the least used server.

For solutions that need a modest boost in read performance, a simple server selection mechanism is all you may need. Larger, more sophisticated solutions may need to build their own load balancer or use a third-party load balancer.[4] Even so, the load balancing mechanism may be entirely application dependent and even in some cases access method dependent (think type of queries).

For hobbyist and enthusiasts IOT solutions and even small to medium-sized commercial IOT solutions, the simple server selection mechanism is more than sufficient. If performance starts to slow, you simply add another server and spread the reads.

You may be wondering where or how you would set up read scaling. One possible implementation is the use of a topology of low-cost computer boards. Another is a cloud service that provides MySQL instances you can create and destroy on the fly. Fortunately, MySQL works in the cloud the same way it does on any other device—the cloud server you get is just a virtualized server instead of real hardware.

---

[3]There are other tricks, but this is the safest way.
[4]Oracle doesn't have a load balancer for MySQL read scaling yet. But it does have a connection router capability in the MySQL Router product.

Now that you understand what read scaling is and how it is used to improve performance, let's see a simple server selection code example you can use to spread reads across several servers. Let's begin with the design.

Overview of the Design

Rather than design a code solution, I present a solution that you can use to implement the persistence component in MySQL. That is, we can use the database server as a gatekeeper to help ensure two clients do not choose the same slave. It may not be the most efficient mechanism available and falls short of being a general-purpose load balancer (but it mimics a rudimentary one), it presents a mechanism that permits you to use it without modifying your code every time you add a new server. In fact, it will select a server automatically from the list of available servers, and you don't even have to create and manage a configuration file.[5]

The key concept is a feature of replication itself. We will use the SHOW SLAVE HOSTS statement on the master to get a list of the slaves. Recall we require the use of the --report-host and --report-port options for all slaves to get the data to appear in the list. We then use a round-robin mechanism to select the next server in the list.

Another key concept is the use of the database server to store the last server ID used. We create a simple database and table with a single row as follows. Notice we store the server ID, hostname, and port.

```
CREATE DATABASE read_scaling;
CREATE TABLE read_scaling.current_slave(server_id INT, host CHAR(30), port INT);
INSERT INTO `read_scaling`.`current_slave` VALUES (0,NULL,0);
```

Notice the INSERT statement. You will need to insert a starting row because the code is designed to update only the single row, making it small and easy to use. However, you may be wondering about that single row in the table. Savvy readers who have worked with applications that have multiple clients may be wondering how to keep two or more clients from colliding and updating the row at the same time (or out of sequence). Here is where the power of the database helps us.

In computer programming jargon, we call the portion of a program that must be executed such that it must complete all at once a *critical section*. In this case, it is similar to a transaction in that vein. MySQL provides a SQL statement named LOCK TABLES that we can use to lock the table, update it, and then release the lock. In this way, we ensure one and only one client can update the table at a time because the WRITE lock blocks all other connections from reading or writing to the table. The following shows the SQL statements we will use to do this in our code:

```
LOCK TABLES read_scaling.current_slave WRITE;
UPDATE read_scaling.current_slave SET server_id=2,host='test123',port=13001;
UNLOCK TABLES;
```

But wait, what about the data—won't it get replicated to the slaves? As a matter of fact, it will. While it really won't matter since we won't be reading the data from the slaves, there is a good reason to allow this data to replicate. Consider what would happen if the master were to go down. If you had not replicated the data, you would lose it when the new master is configured; you would have to set it up again. But that's not really a problem given its simplicity. So, it is fine to allow it to replicate.

---

[5]Not that there is anything wrong with configuration files, but if you've ever encountered an installation where there are multiple configuration files scattered across several parts of the solution, you'll grow to appreciate not requiring them.

WHAT I DON'T WANT CERTAIN DATA REPLICATED?

Sometimes there may be data you do not want replicated. In this case, you can use the replication filters built into MySQL to tell the server to not replicate certain data. There are filters available on the master (binary log options) and the slaves (replication options). They can be inclusive or exclusive.

Setting a filter on the master ensures the data matching the filter is not saved in the binary log and thus never transmitted to the slaves. Setting a filter on the slave ensures that, while the data is being transmitted, it is discarded on the slave. Think of it this way, if you want some data to never be replicated, use the master filters. If you want one or more slaves but not all to not get the data, use slave filters.

Binary log filters include `--binlog-do-db` and `--binlog-ignore-db`. Replication filters include `--replicate-do-db`, `--replicate-ignore-db`, `--replicate-do-table`, and `--replicate-ignore-table`. See the online MySQL reference manual for more information about setting and maintaining the filters along with pitfalls for using them (http://dev.mysql.com/doc/refman/5.7/en/replication-options-binary-log.html) and (http://dev.mysql.com/doc/refman/5.7/en/replication-options-slave.html).

Finally, the code will be written as a class that you can place in your library and include in any client code you want. For the purposes of demonstrating the code, I will include test code, but you can remove this and use the class in a library. The class is named `select_read_server` and is written in Python but could easily be written in whatever language you choose.

Write the Code

The class needs only a single public method named `get_next_server()` that returns a dictionary of the server ID, host, and port. There are a number of helper methods needed, which will be made private (designated by starting with an underscore character in the name).

We need a method to connect to the master and retrieve a list of all the slaves and store that information in memory. We name this method `_get_server_list()`. Next, we need a method to record the server ID of the slave chosen so that any other clients can read it. We name this method `_set_server_id()`. Recall we will use the database to store the server ID chosen and will use locking to prevent other clients from interrupting the update to the table. Thus, `_set_server_id()` must execute the locking mechanism described earlier. We also need a method to read the server ID from the table. We name this method `_get_current_server_id()`. Finally, we include a simple loop to demonstrate how to use the class to retrieve the information for the next slave. Listing 8-4 shows the complete code.

Listing 8-4. Simple Read Scaling Server Selector

```
# Demonstration of a simple round robin read scaling slave selector
#
# Use this class in your own application to choose the next slave
# in the topology.
#
# Note: you must have a database and table setup on the master as
# follows:
#
```

```python
# CREATE DATABASE read_scaling;
# CREATE TABLE read_scaling.current_slave(server_id INT, host CHAR(30), port INT);
#
# Note: You must also have a replication topology of a master and at least
#       two slaves for the demo output to be meaningful. In fact, running this
#       on a master with no slaves will result in an error.
#
from __future__ import print_function
import mysql.connector as mysql
import operator
import sys
import time

LOCK_TABLE = "LOCK TABLES read_scaling.current_slave WRITE"
SET_SERVER = "UPDATE read_scaling.current_slave SET server_id={0},host='{1}',port={2}"
UNLOCK_TABLE = "UNLOCK TABLES"
GET_CURRENT_SERVER = "SELECT server_id FROM read_scaling.current_slave"

master = {
  'user': 'root',
  'password': 'root',
  'host': 'localhost',
  'port': 13001,
}

class select_read_server(object):
    def __init__(self):
        self.servers = []
        self.cur_server_id = -1;

    # Get the list of servers from the master and save the host,
    # port, and server id.
    def _get_server_list(self):
        conn = mysql.connect(**master)
        cur = conn.cursor()
        cur.execute("SHOW SLAVE HOSTS")
        # Save only the id, host, and port
        for row in cur.fetchall():
            server = {
                'id': row[0],
                'host': row[1],
                'port': row[2],
            }
            self.servers.append(server)
        # order the servers by server_id
        cur.close()
        conn.close()
        self.servers.sort(key=operator.itemgetter('id'))

    # Set the server in the database
    def _set_server(self, id, host, port):
```

```python
            self.cur_server_id = id
            conn = mysql.connect(**master)
            cur = conn.cursor()
            cur.execute(LOCK_TABLE)
            query = SET_SERVER.format(id, host, port)
            cur.execute(query)
            print(">", query)
            cur.execute(UNLOCK_TABLE)
            cur.close()
            conn.close()

    # Get the current server id from the database
    def _get_current_server_id(self):
        # if first time, skip
        if self.cur_server_id == -1: return
        conn = mysql.connect(**master)
        cur = conn.cursor()
        cur.execute(GET_CURRENT_SERVER)
        self.cur_server_id = cur.fetchall()[0][0]
        cur.close()
        conn.close()

    # Get the next server in the list based on server id.
    def get_next_server(self):
        self.servers = []
        self._get_server_list()    # update the server list
        if not self.servers:
          raise RuntimeError("You must have slaves connected to use this code.")
        self._get_current_server_id()  # get current server in the database
        for server in self.servers:
            if server["id"] > self.cur_server_id:
                # store the current server_id
                self._set_server(server["id"], server["host"], server["port"])
                return server
        # if we get here, we've looped through all rows so choose first one
        server = self.servers[0]
        self._set_server(server["id"], server["host"], server["port"])
        # return the current server information
        return server

# instantiate the class - only need this once.
read_server_selector = select_read_server();

# demonstrate how to retrieve the next server and round robin selection
for i in range(0,10):
    print(i, "next read server =", read_server_selector.get_next_server())
    sys.stdout.flush()
    time.sleep(1)
```

Notice we have the connection information for the master that we use to get the slave information. Also, notice that we get this list each time the next server is requested. In this way, the code will keep the list updated so that if a server is added or removed (and it is dropped by the master), the connection list will update automatically.

However, the output of the SHOW SLAVE HOSTS statement is unordered and may not appear in server ID order. Thus, I added a sort method to ensure the round-robin mechanism loops through the server IDs in ascending order.

■ **Tip** It can take a few seconds for the slave hosts view to update if a slave disconnects unexpectedly. So, you may want to consider adding code to check that the slave is still connected if you plan to use this code in a production environment.

Test the Sketch

Now let's test the code. All you need to do to run it on your system is to change the master information to match your setup and execute the code with the python command. Listing 8-5 shows a sample run. I added code to print out the results from the class as well as printing the SQL statement used to update the table so that you can see it is indeed selecting the next server in the list and repeating the loop starting from the first server ID in the list.

Listing 8-5. Testing the Read Scaling Selector

```
$ python ./read_scaling_demo.py
> UPDATE read_scaling.current_slave SET server_id=2,host='localhost',port=13002
0 next read server = {'host': u'localhost', 'id': 2, 'port': 13002}
> UPDATE read_scaling.current_slave SET server_id=3,host='localhost',port=13003
1 next read server = {'host': u'localhost', 'id': 3, 'port': 13003}
> UPDATE read_scaling.current_slave SET server_id=4,host='localhost',port=13004
2 next read server = {'host': u'localhost', 'id': 4, 'port': 13004}
> UPDATE read_scaling.current_slave SET server_id=2,host='localhost',port=13002
3 next read server = {'host': u'localhost', 'id': 2, 'port': 13002}
> UPDATE read_scaling.current_slave SET server_id=3,host='localhost',port=13003
4 next read server = {'host': u'localhost', 'id': 3, 'port': 13003}
> UPDATE read_scaling.current_slave SET server_id=4,host='localhost',port=13004
5 next read server = {'host': u'localhost', 'id': 4, 'port': 13004}
> UPDATE read_scaling.current_slave SET server_id=2,host='localhost',port=13002
6 next read server = {'host': u'localhost', 'id': 2, 'port': 13002}
> UPDATE read_scaling.current_slave SET server_id=3,host='localhost',port=13003
7 next read server = {'host': u'localhost', 'id': 3, 'port': 13003}
> UPDATE read_scaling.current_slave SET server_id=4,host='localhost',port=13004
8 next read server = {'host': u'localhost', 'id': 4, 'port': 13004}
> UPDATE read_scaling.current_slave SET server_id=2,host='localhost',port=13002
9 next read server = {'host': u'localhost', 'id': 2, 'port': 13002}
```

Notice we call the method to get the next server ten times. Notice also that the code successfully loops through the slaves in a round-robin selection updating the database with each new selection. Cool, eh?

High Availability IOT Nodes

You have seen a number of techniques using MySQL replication that you can leverage to achieve a level of high availability (reliability) in your IOT solution. But these cover the database availability and not the underlying hardware for your IOT nodes. This section examines two examples for how to achieve high availability in your IOT nodes.

You will see an example of how to make a redundant and automatic recovery of a data collector failure as well as a data aggregator that can cache data should its connection to the database fail. The examples use Arduino microcontrollers, but the concepts can be applied to other microcontrollers, and thus you should be able to duplicate these concepts in your own solutions.

Example: Redundant Data Collectors

One thing that you will eventually encounter if you work with microcontrollers long enough on increasingly complex IOT solutions is that sometimes the microcontroller will go wonky[6] or completely wig out.[7] This could be because of a power failure, unclean power supply, tampering, poor programming, memory corruption, or just plain fate. I've had microcontrollers that have run seemingly forever and others that turn hinky within a few days. Whatever the case, microcontrollers can and do indeed occasionally fail.

One way to combat this and to ensure our solution is more reliable (and high available to a degree) is to build redundancy into our hardware. We can do this using a master/slave setup of two microcontrollers. Both microcontrollers would have the same type of sensors (if not the same sensors[8]), but only the master would be writing the data to the database. Furthermore, we want to make a data collector redundant with automatic failover to the slave so that if the master fails, the slave takes over and we do not lose any significant data.

Yes, we are talking about creating a simple replication protocol for the master and slave to communicate. Let's consider how we would do this.

Overview of the Design

There are many ways we could implement a simple protocol between the master and slave. In this case, we'll keep it simple and use the concept of a heartbeat. This is a simple message transmitted by one server and acknowledged on another. The idea is if the heartbeat fails or is missed after a significant time period, the second server can assume the first server has failed or gone wonky, has gone hinky, or has wigged out entirely.

Fortunately, we can do this with a capability built into the Arduino: the I2C (pronounced "eye-two-see") interface bus with the wire protocol. The I2C bus uses two pins or wire (sometimes called *two-wire*): the Serial Data Line (SDA on pin 4) and Serial Clock Line (SCL on pin 5). The interface allows for multiple devices to connect by registering a device number when initiated. Indeed, there are a great number of devices including sensors that use the I2C interface.

■ **Caution**   If you are using a Uno, Leonardo, or similar board with the newer header layout, the SDA and SCL pins are located above pin 13 (D2 and D3). Similarly, larger boards like the Mega or Due have the I2C interface in a different location (D20 and D21). Be sure to check the header layout of your board; otherwise, you could be faced with a frustrating debugging session.[9]

---

[6]Not to be confused with hinky, which is worse.

[7]A highly technical description of erratic behavior.

[8]Not recommended, but I've seen it work well in experiments. I would use duplicate sensors.
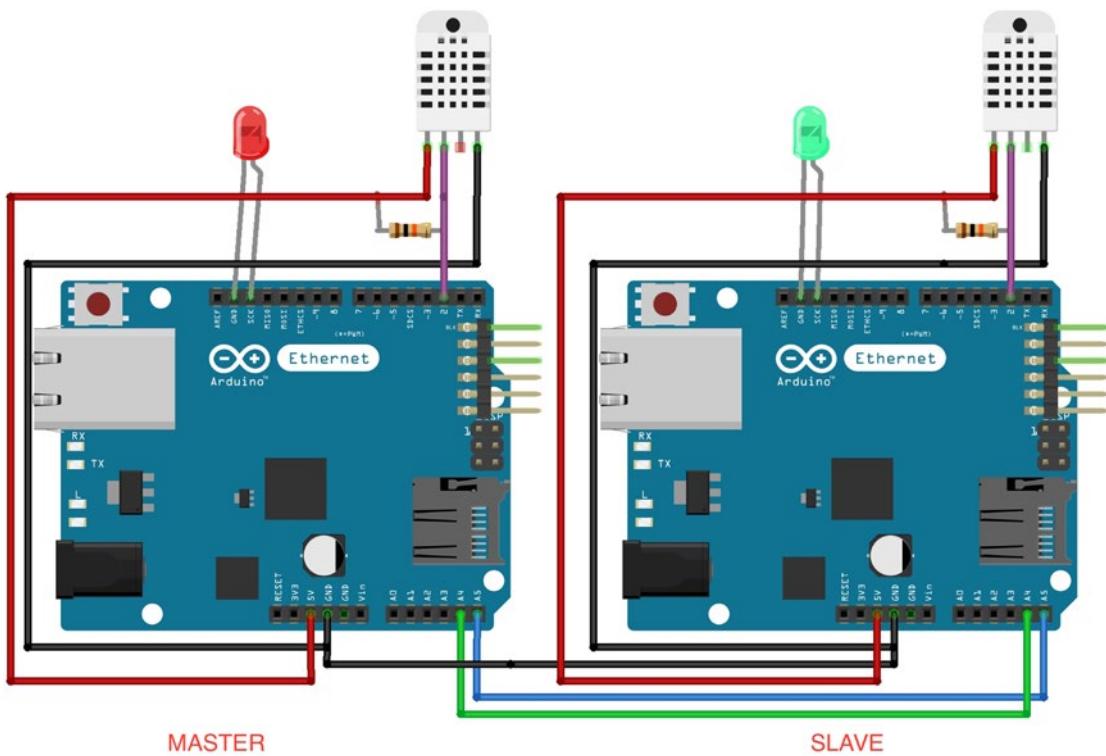
[9]Can you guess how I know this? Yep. I was ready to skeet shoot my Leonardo until I remembered the pin layout was different. Use enough Arduino boards, and you may find you'll want to do the same!

On the I2C bus, one device is designated as a write and the other a reader. The code therefore uses the master as a writer and the slave as a reader, passing a simple message from the master to the slave. We will use this configuration to construct a simple heartbeat algorithm whereby the master sends a short message at a specific interval. This is sometimes called a *pulse* to complete the simile.

Thus, we can detect when the master is no longer communicating when the heartbeat does not occur for a span of pulses. This can be detected by allowing a certain amount of time to expire without a pulse or heartbeat message. We can use this event to trigger the failover to the slave for data collection. More specifically, a timeout is used on the slave to wait for a certain amount of time to pass before declaring the master dead. If the master does not send its message after a predetermined time, the slave takes over and starts reading data and saving it to the database server. Thus, we can create a simple automatic failover using redundant data collectors.

Assemble the Hardware

The hardware for this example requires two Arduino boards wired together via the I2C bus. We add an LED to each for visual confirmation of messages sent and received. I used a red LED on the master and green LED on the slave, but use whatever you have on hand—color is not important. Figure 8-4 shows how to wire the hardware.
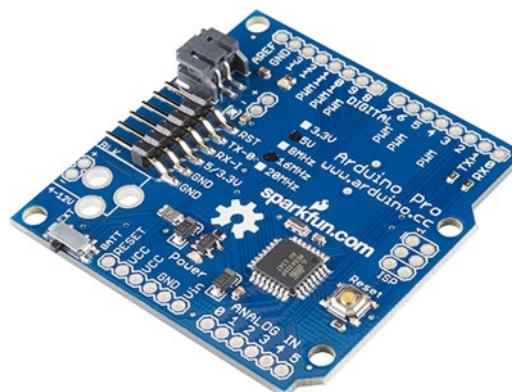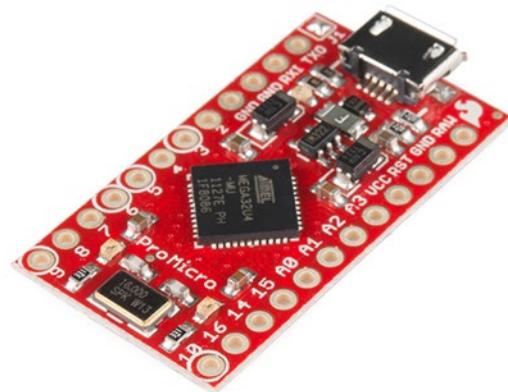
Figure 8-4. *Redundant data collector (master/slave)*
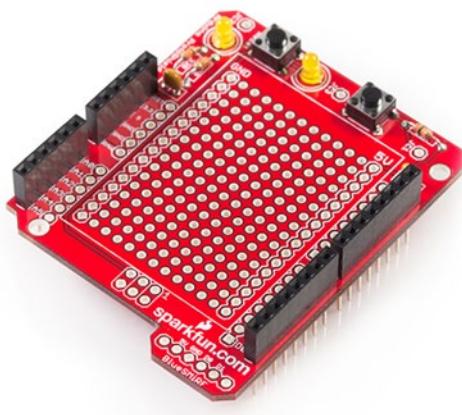
■ **Tip**   If you want to power the Arduino boards with a single power supply, you can wire the 5V pin on one Arduino to the VIN pin on the other. However, this will not protect you against power supply issues, so I do not recommend this for production use.

Notice the figure shows a sensor added. This is for illustration purposes; you do not need it to execute the example code.

STANDARD HEADERS AND WIRING

By now you should be wondering how to get those wires to stay in the pin headers. The standard headers for Arduino boards are great for prototyping but not so much for installation of final solutions. Fortunately, you can get Arduino boards without the headers, and indeed some clone Arduino boards are designed with special layouts, making them easy to use with breadboards or to hardwire circuits. Alternatively, you can use a prototype shield to build your circuit. The following figures depict two alternative Arduino boards and a prototyping shield that you can use in permanent solutions such as soldering wires to the board.

Images courtesy of sparkfun.com.

Write the Sketch

The example code is simplified to make it easier to experiment with. More specifically, I do not include any code for reading sensors or writing to the database. I have left comments for where you would add these elements. Recall, we always want to start with skeleton code like this when developing complex solutions. If you lumped everything in one code file and something wasn't working, you may not be able to easily discover which part is causing the problem (or even if multiple parts are failing).

I also use a trick of programming called *condition compilation*. This allows me to have sections of the code that can be used (compiled) one way but discarded if used (compiled) another. In this case, I wanted one source file but conditionally use some parts for the master and other parts for the slave.

To mark sections as conditional, we use the #if defined clause and its opposite #if not defined clause along with a #define to determine which code is included when we compile and upload. Thus, in the following code, you will see a #define SLAVE that you can comment out and cause the code to compile for the master. Again, leave #define SLAVE and the code is compiled as the slave; comment it out and you've got the master.

Rather than throw all the code at you at once, let's go through each piece and discuss what needs to be done starting with the setup() code for each master and slave.

The code needed for the master and slave communication comes from the wire library. Both the master and slave must initiate the wire protocol with wire.begin(), but the slave needs to register the device number and provide a callback (a special method that is called when an event is trigger, in this case a message received from the master). Both master and slave need to set up a PIN for the LED and start the serial class. Thus, the setup() code uses the conditional compilation as discussed earlier. The code is shown next. Notice how the code for the slave does more than the master and we have a section that is not in the conditional compilation block. This code is compiled for either master or slave.

```
void setup() {
#if defined SLAVE
  Wire.begin(address);         // join i2c bus
  Wire.onReceive(getHeartbeat); // register the heartbeat code
  startTime = millis();        // initiate timer
```

```
    elapsedTime = 0;
#else
    Wire.begin();
#endif
    pinMode(13, OUTPUT);              // turn on onboard LED (pin 13)
    Serial.begin(115200);
    // TODO: Add any database initialization code here
}
```

So, what is that callback for the slave? Here we simply receive a message from the master. The code to do this is shown next and is pretty self-explanatory. Notice we read the characters from the master and print them out to the serial monitor. We really don't care what message the master sends, just so long as it sends something. Notice too we set up the timer to start over whenever the master has sent a message.

```
void getHeartbeat(int chars) {
    char ch;
    while (Wire.available()) {
        ch = Wire.read();
    }
    Serial.println("Master is ok.");
    // Reset timer since master is Ok
    startTime = millis();
    elapsedTime = 0;
    blink();  // visual feedback
}
```

On the master, we send the message with the following code. As you can see, we just write a simple, short message. However, we cannot broadcast a message on the I2C bus; it must be directed to a specific device. Here we use a simple variable, address, to set the address. Since the code is from the same file, we ensure both master and slave are compiled with the same value.

```
void sendHeartbeat() {
    Wire.beginTransmission(address);
    Wire.write("hello!");
    Wire.endTransmission();
    blink();  // visual feedback
}
```

I should note at this point that we don't want code specifically for the slave to be included with the master (it will just waste precious memory), so you will see the conditional compilation block around the slave-specific code.

So, how does the slave know the master is dead? We use the timeout code designated with the elapsedTime, startTime, and maxTime variables. That is, we record the number of milliseconds from the Arduino with millis()[10] each time through our loop so that if the time elapsed is more than the maximum time set, we declare the master dead. In this case, we write the sketch (code) for the slave to execute the same code the master does to read the sensor and save the data to the database.

---

[10]A rudimentary albeit slightly inaccurate way to measure time without a real-time clock but good enough for use as a rough timer.

All that is left now are the details of communicating with the serial monitor, timer expiration, and blinking the LED—all of which should be familiar. In any case, Listing 8-6 shows the completed code for the redundant data collector with automatic failover.

Listing 8-6. Redundant Data Collector with Failover

```
/**
  Example Arduino redundant data collector

  This project demonstrates how to create a data
  collector master/slave for redundancy. If the master
  fails, the slave will take over.

  The sketch has both the master and slave code with the
  default the slave role. To use the sketch as a master,
  comment out the #define SLAVE.
*/
#include <Wire.h>

// TODO: Add MySQL Connector/Arduino include, variables here

#define SLAVE

int address = 8;                // Address on I2C bus

// Blink the LED on pin 13
void blink() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
}

// Record data to database
// Stubbed method for writing data
void recordData() {
  // TODO: Complete this code for your sensor read and the write to MySQL.
}

#if defined SLAVE
unsigned long startTime;        // start of simple timer
unsigned long elapsedTime;      // number of milliseconds elapsed
unsigned long maxTime = 10000;  // timeout value (10 seconds)

// Get the heartbeat message from the master
void getHeartbeat(int chars) {
  char ch;
  while (Wire.available()) {
    ch = Wire.read();
  }
```

```
  Serial.println("Master is ok.");
  // Reset timer since master is Ok
  startTime = millis();
  elapsedTime = 0;
  blink();  // visual feedback
}

#else
// Send the heartbeat pulse
void sendHeartbeat() {
  Wire.beginTransmission(address);
  Wire.write("hello!");
  Wire.endTransmission();
  blink();  // visual feedback
}

#endif

void setup() {
#if defined SLAVE
  Wire.begin(address);            // join i2c bus
  Wire.onReceive(getHeartbeat);   // register the heartbeat code
  startTime = millis();           // initiate timer
  elapsedTime = 0;
#else
  Wire.begin();
#endif
  pinMode(13, OUTPUT);            // turn on onboard LED (pin 13)
  Serial.begin(115200);
  // TODO: Add any database initialization code here
}

void loop() {
#if defined SLAVE
  Serial.println("looping...");
  // Check timeout
  elapsedTime += millis() - startTime;
  Serial.println(elapsedTime);
  if (elapsedTime > maxTime) {
    // Master is dead. Read and save the sensor data!
    Serial.println("Master has died! Oh, my...");
    // OPTIONAL: reset master timer to try again.
    startTime = millis();
    elapsedTime = 0;
```

```
    // Record data since master is dead.
    recordData();
  }
#else
  recordData(); // read and save data
  Serial.print("Sending heartbeat...");
  sendHeartbeat();
  Serial.println("done.");
#endif
  delay(3000);  // wait 3 seconds
}
```

Don't be too overwhelmed with this code. I know it is rather complex and perhaps more complex since it uses conditional compilation, but it saves us from having two projects instead of one. And given that the master and slave need the same code to talk with a MySQL server and sensor(s), we can write it once and save some headaches of debugging and maintaining duplicate code.
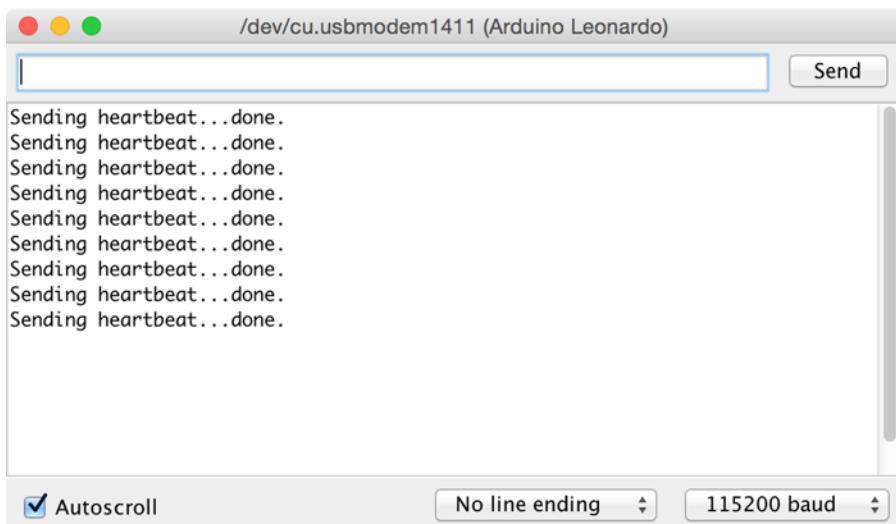
Take some time to read through this code before you try to run it yourself. It is also a good idea to double-check and triple-check your wiring before moving on to running the code, which can be a bit tricky, so read the next section carefully.

Test the Sketch

Testing the sketch can be a bit intimidating, so let's go over what we need to do. First, we must compile the sketch and upload it to the master. We simply comment out the #define, select the correct board and port, and then upload the sketch. Next, we remove the comment from the #define, select the correct board and port for the slave, and then compile and upload.
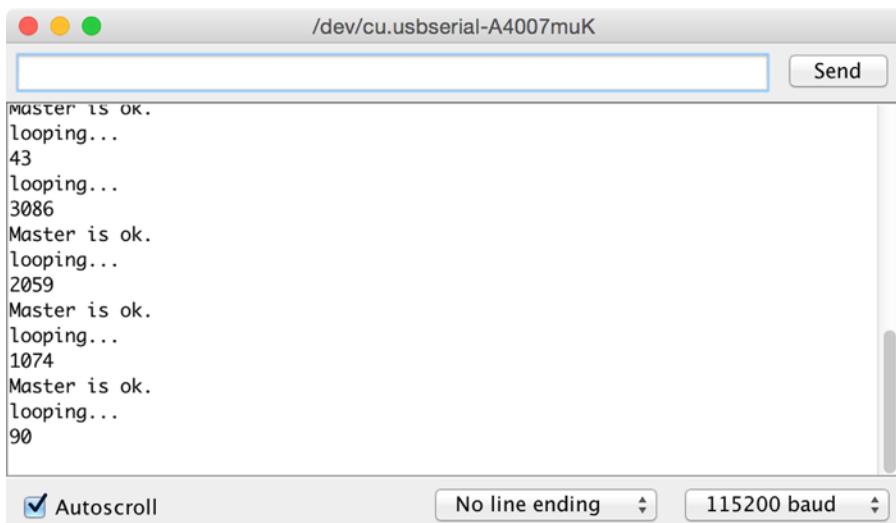
What I did (and I highly recommend this) is to have both the master and slave connected to your computer via two USB connections. This will make setting up the experiment much easier. Indeed, you can switch between the two simply by setting the port in the Arduino IDE and starting the serial monitor. Note, however, that the Arduino will force the Arduino to reset when the serial monitor connects. This is fine for testing, but just be aware it means it may take a few seconds to see the master/slave code working.

When connected to the master via the serial monitor, you should see output similar to that shown in Figure 8-5. While it is not interesting, it can help you determine whether your master is sending the message. Don't forget we also wired in an LED, so you can watch the LED, and it should blink along with each message feedback you see.

Figure 8-5. *Sample debugging output (master)*

Next, when you connect to the slave, you should see messages that depict whether the slave has received the message (heartbeat) along with a counter of the elapsed time. Should the master fail, you will also see a message stating that. Figure 8-6 shows sample output from the slave.

Figure 8-6. *Sample debugging output (slave)*

Don't despair or give up if you encounter problems. Go back and check your wiring and make sure everything is wired correctly. Once you are satisfied that isn't a problem, compile and upload the sketches again, making sure you are using the correct board and port for each master and slave. Uploading the slave (or master) sketch to both boards is easy to do by accident. In this case, it may have been easier to have two separate sketches, which you can do if you are having problems keeping the #define straight.

You should also consider uploading the master sketch and leaving it running while you upload the slave sketch and then connect to the serial monitor output for the slave. I found this sometimes helps keep things straight. After all, the majority of the tricky code is on the slave.

Finally, make sure there are no power issues or similar power incompatibilities between the master and slave boards. I tested using a Leonardo and a much older board and everything worked fine. However, I have seen where some boards, particularly clone boards, do not always work correctly with some I2C devices, and since we're using the slave as an I2C device, that could be a problem source.

■ **Tip** Be sure you have the `#define` commented out to compile the sketch for the master.

Once you start seeing the correct output, celebrate that it worked! This sketch—however small—implements a sophisticated heartbeat and automatic failover, which is something other solutions with more powerful hardware may not be capable of achieving. However, as you can see, with a little ingenuity you can do amazing things with simple hardware like a microcontroller such as the Arduino.

Now the real test begins. Unplug the master and slave from your laptop and power them up. I would power on the master first and then the slave. Does the LED on the master blink followed by the LED on the slave? As long as these LEDs are flashing, the master heartbeat is being sent and received. Now you see why I added LEDs—to test the solution without a computer. If you do not see this happen, try resetting or powering off the boards and then back on again. I have found sometimes the I2C bus doesn't get going quite right and rebooting seems to help.

Once you are satisfied everything is working, you can add the code to read the sensor and write the data to the MySQL database and then deploy your redundant data collector with failover in your IOT solution. Remember that smaller boards may not have the memory needed to support multiple, large libraries, so you may need to use a larger Arduino board for your own solution. Just remember to find the SDA and SCL pins and wire them correctly.

Now let's see how to handle faults when writing data to the database.

Example: Fault-Tolerant Data Collector

Achieving fault tolerance with MySQL is not overly difficult and, as you have seen, can be accomplished using MySQL replication for redundancy, recovery, scalability, and ultimately high availability for your database component. However, achieving fault tolerance on a typical microcontroller-based node that writes data to the database can be a bit more difficult.

In this section, I present a project you can build yourself. The project is a simple plant soil moisture data collector with fault detection and recovery for writing data to the database. You already saw such a project earlier in the book, but this time we're adding the fault detection and recovery mechanisms, which, as you will see, adds quite a bit more complexity to the code and even requires additional hardware!

If you consider a data collector node in its most basic functionality, it should read data from one or more sensors and then pass that data on to a data node or database, or even store the data locally to a file. So, what possible faults can we detect? Clearly, writing code to detect that the node itself is kaput is not feasible (at least not without a redundant node). So, we consider only those faults that a data collector could recover from: failure reading the sensor(s) and failure to store data.

For this project, we focus on only one of those faults and specifically the connection to the database server. In this case, we are concerned about the node becoming isolated on the network, the database server going down, or any other form of interruption to communication from the data collector to the database. I chose this particular fault to demonstrate the added complexity needed. As you will see, it is not trivial, but it isn't overly complicated either—it just takes a bit of work.

Let's begin with a brief description of the design and then a list of components (hardware) needed.

Overview of the Design

The goal is to make the node fault tolerant of the database connection. More specifically, should the database become unreachable, we want to store the data locally (caching the data) to the SD card and then, once the database is reachable, save that cached data to the database.
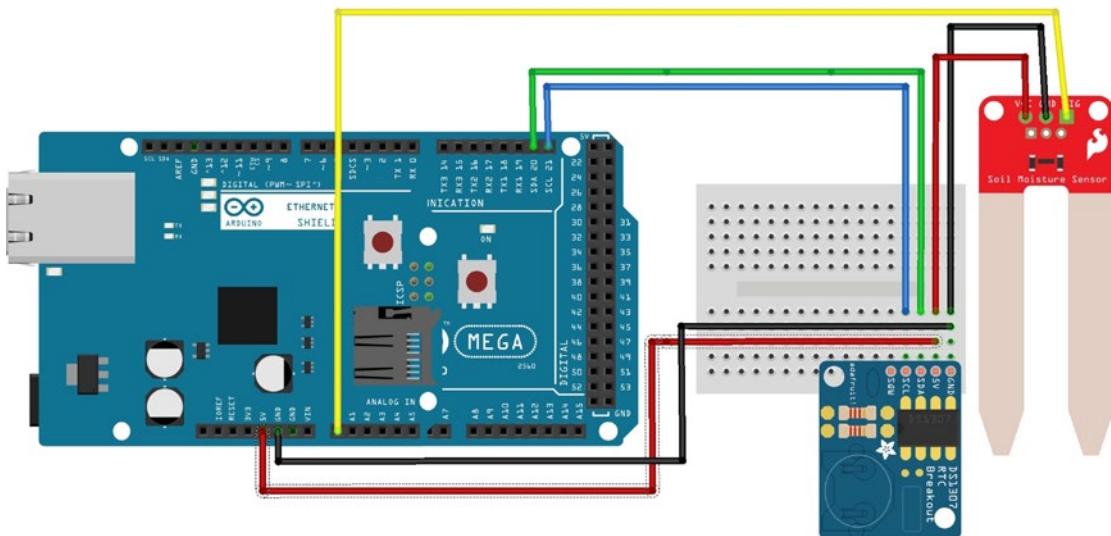
The database is a simple affair with a table designed to capture only the soil moisture sensor value and the data and time the value was read (stored). The following shows the SQL statements needed to create the sample database for this project:

```
CREATE DATABASE `plant_moisture`;
CREATE TABLE `plant_moisture`.`plants` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `moisture_raw` int(11) DEFAULT NULL,
  `date_saved` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=257 DEFAULT CHARSET=latin1;
```

Notice we create a simple table with an auto-increment column, a column for the soil moisture, and a date-stored column. I forgo the more complicated annotation for the soil moisture for simplicity, but feel free to add that column and trigger as an exercise.

Assemble the Hardware

The hardware for this project will be an Arduino, a soil moisture sensor, an Ethernet shield, and a real-time clock module. Figure 8-7 shows how to wire up everything.

Figure 8-7. *Arduino Mega with Ethernet shield, RTC module, and moisture sensor*

But wait! Why are we using an Arduino Mega? This is one of those times that we see just how resource-intensive fault detection and recovery can be. Recall the Arduino Uno and similar small boards have a finite amount of memory. We are adding a host of libraries: the Connector/Arduino for connecting to the MySQL server, the SD card for caching the data, and the RTC module to read the date and time. Recall also the Connector/Arduino requires several other libraries. This, combined with the code to detect the fault and recover from it, we will easily exceed the memory of an Uno board. Thus, we use the Mega board, which has more than sufficient memory. As you will discover in your own projects (if not now, soon), writing complex code for the Arduino often requires stepping up to the larger boards.

You may also be wondering about the RTC module. Recall our database has a timestamp column for storing the data and time the sample was stored. Recall also that the database server will fill this value in for you. However, if we detect a fault and it is some time (perhaps hours) later before we can reconnect to the database server, those values for the date stored will be incorrect—the database server will use the date and time of the write to the database, not when the sample was taken. Thus, we need the RTC module to get the current date and time to store in the cache along with the value stored. As you can see, caching the data is not just a simple write and read later event!

Let's go over the connections since there are many. First, the soil moisture sensor is wired with 5V and ground coming from the Arduino. I also connect the signal (data) line from the sensor to analog pin 0 on the Arduino. So far, we're good. However, the RTC is not wired the same as it was in the earlier example. This is because the pins for SDA and SCL are in a different place on the Mega. In this case, they're on pins 20 and 21 respectfully. So, we wire the corresponding RTC module SDA and SCL connections to pins 20 and 21 and then get 5V and ground from the Arduino. I use a breadboard in the drawing to make the power and ground connections easier.

WHY NOT USE A LOW-COST COMPUTER BOARD?

One of the goals of presenting this project is to demonstrate how to achieve some rather sophisticated features from a small microcontroller. While the Arduino Mega is a bit more expensive than a Raspberry Pi, learning how to implement fault detection and recovery at the lowest level will help you implement similar and even more sophisticated mechanisms in more powerful boards. If fact, you can implement this project in Python using Connector/Python on a Raspberry Pi and achieve the same level of redundancy. I encourage you to explore this as a challenge.

Now let's look at the code needed for our sketch.

Write the Sketch

There are several parts to the sketch (code). We know we will write a data collector to read samples from a sensor and write the data to the database. Also, should the connection not be viable, we will save data to a local file. When the database connection is reestablished, we must read the data written to the file and insert it into the database. Before we get into the code, we should consider the design goals and determine the key requirements. I've found it helps to write these down even if they are clear in your mind. Indeed, I've found listing the requirements can help you plan and implement the code in more organized manner. I list the major requirements here:

- All data collected should be saved to the database.

- The database connection should be tested to ensure the server is connected.

- In the event the database cannot be connected, data should be stored to a local file.

- Data saved to the local file should be written to the onboard SD card.

- When the database is connected and there is data in the local file, read the data and insert it into the database.

- Cached data should store the correct date and time for the sample collection.

Let's examine each of these requirements and determine what the code should be for each. But first, let's consider a high-level algorithm we can use to encapsulate the mechanism. The following is a high-level overview of the loop() method, that is, the "main" procedure that drives the sketch. We'll see the setup in Listing 8-7 later in this section. I find writing an algorithm like this helpful in organizing your sketch. Not only does it help identify major methods needed, but by keeping the code at a high level, it provides a template for the code in the loop() method that is easy to read and can help verify that the requirements are identified in the code.

```
read_sample()
if <database connection fails> then
  cache_data()  // save data to the file
else
  dump_cache()  // read data from file and insert into database
  write_data()  // save current sample
end if
```

So, now we see that we will need three major methods along with some means to detect whether the database connection has failed. Let's go through each of these and discuss them beginning with reading the sample. Since we are using an analog sensor, we can read the sensor with the analogRead() method, as shown here:

```
value = analogRead(sensorPin);
```

Next, let's consider the requirement to detect when the connection to the database fails. One way to do this is to use the connected() method in the Connector/Arduino library, but this will tell you only if the connection opened previously is still open. It should be noted that even if the database server goes offline, it could take some time for the connected() method to return false (connection lost).

A better mechanism is to attempt a connect() each time through the loop and then close() after the data is saved. The connect() method returns much more quickly should the server be unreachable. Now when we use the connected() method, it returns immediately that the database server is not connected. The following shows an excerpt of how to set this up in the code:

```
// Attempt to connect to the database server
if (!conn.connected()) {
  if (conn.connect(server_addr, 3306, user, password)) {
  ...
  }
}
...
conn.close();
```

You have already seen an example of how to write data to a file on the SD card. Recall it requires the SD library and code to open a file, write the data, and then close the file. The only tricky part is we must get the date and time string and write that to the file, preserving when the sample was taken. You already saw how to read date and time from the RTC, so that code is also familiar.

You have also seen how to read data from the file, but not in a way that resulted in inserting the data in the database. But mechanically, this insert is the same as the write_data() method primitive shown earlier. Thus, we can reuse this method (write it once, use it multiple places). The following shows the code for the method:

```
bool write_data(int val, const char *dateStr) {
  String queryStr = String("INSERT INTO plant_moisture.plants VALUES (NULL,");

  queryStr += String(val);
  queryStr += ",";
  if (dateStr != "NULL") queryStr += "'";  // quote string for values
  queryStr += dateStr;
  if (dateStr != "NULL") queryStr += "'";  // quote string for values
  queryStr += ")";
  // write the data here
  Serial.println(queryStr);
  // Create an instance of the cursor passing in the connection
  MySQL_Cursor *cur = new MySQL_Cursor(&conn);
  cur->execute(queryStr.c_str());
  delete cur;
}
```

Notice here I've written the code to detect whether the date string passed is NULL (a character string, not the null concept or even 0 or \0 terminators). If the string is not NULL, the string is a date and time value, and we must place quotes around it in the resulting INSERT statement.

That's it. We have all the methods from the high-level algorithm covered. We need only implement the required startup code for the RTC, SD card, and MySQL Connector/Arduino. Listing 8-7 shows the complete code for this sketch.

Listing 8-7. Fault-Tolerant Data Collector

```
/**
  Example Arduino fault tolerant data collector

  This project demonstrates how to save data to a
  microSD card as a cache should the node fail to
  save data to a MySQL database server.
*/
#include <Wire.h>
#include <SD.h>
#include <SPI.h>
#include <Ethernet.h>
#include <MySQL_Connection.h>
#include <MySQL_Cursor.h>
#include "RTClib.h"

// Pin assignment for Arduino Ethernet shield
#define SD_PIN 4
// Pin assignment for Sparkfun microSD shield
//#define SD_PIN 8
// Pin assignment for Adafruit Data Logging shield
//#define SD_PIN 10
```

```
#define LOGFILE "cache.txt"          // name of log file for caching data

int sensorPin = 0;                   // Pin for reading the sensor

RTC_DS1307 rtc;                      // real time clock

byte mac_addr[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress server_addr(10,0,1,8);    // Supply the IP of the MySQL *server* here

char user[] = "root";                // can be anything but the user must have
char password[] = "secret";          // access rights to connect (host must permit it)

EthernetClient client;
MySQL_Connection conn((Client *)&client);

// Get the date and time from the RTC
String get_datetime() {
  DateTime now = rtc.now();
  String dateStr = String(now.year());
  dateStr += "-";
  dateStr += now.month();
  dateStr += "-";
  dateStr += now.day();
  dateStr += " ";
  dateStr += String(now.hour());
  dateStr += ":";
  dateStr += String(now.minute());
  dateStr += ":";
  dateStr += String(now.second());
  return dateStr;
}

// Save the data to the SD card for later processing
// Note: What happens when the disk is full?
bool cache_data(int value) {
  File log_file;

  String row = String(value);
  log_file = SD.open(LOGFILE, FILE_WRITE);
  if (log_file) {
    // Get datetime
    row += ",";
    row += get_datetime();
    // Save data to the file
    log_file.println(row);
    log_file.close();
    Serial.println("Data cached to log file.");
    return true;
  }
```

```
  Serial.println("ERROR: Cannot open file for writing.");
  return false;
}

// Save the data to the database
bool write_data(int val, const char *dateStr) {
  String queryStr = String("INSERT INTO plant_moisture.plants VALUES (NULL,");

  queryStr += String(val);
  queryStr += ",";
  if (dateStr != "NULL") queryStr += "'";  // quote string for values
  queryStr += dateStr;
  if (dateStr != "NULL") queryStr += "'";  // quote string for values
  queryStr += ")";
  // write the data here
  Serial.println(queryStr);
// Create an instance of the cursor passing in the connection
  MySQL_Cursor *cur = new MySQL_Cursor(&conn);
  cur->execute(queryStr.c_str());
  delete cur;
}

// Read the cache file on disk and write rows found to the database
bool dump_cache() {
  File log_file;
  char char_read;
  char buffer[128];
  int i = 0;
  int comma_pos = 0;
  int val = 0;

  // if no file, no cache!
  if (!SD.exists(LOGFILE)) return true;
  log_file = SD.open(LOGFILE);
  if (log_file) {
    // Read one row at a time.
    while (log_file.available()) {
      char_read = log_file.read();
      if (char_read != '\n') {
        // look for fist value
        if (char_read == ',') {
          buffer[i] = '\0';
          val = atoi(buffer);
          i = 0;
        } else {
          buffer[i] = char_read;
          i++;
        }
      } else {
        buffer[i] = '\0';
        i = 0;
```

```
      // Write the row to the database
      write_data(val, buffer);
    }
  }
  // close the file:
  log_file.close();
  // Now remove the file
  if (SD.remove(LOGFILE)) {
    Serial.println("Cache emptied. File removed.");
  }
  return true;
}
// if the file didn't open, print an error:
Serial.println("ERROR: Cannot open file for reading");
return false;
}

void setup() {

  Serial.begin(115200);
  while (!Serial); // wait for serial to load

  // Turn on the SPI for the SD card (Arduino Ethernet)
  // Note: required for Arduino Mega!
  pinMode(10,OUTPUT);
  digitalWrite(10,HIGH);

  // Start the RTC
  rtc.begin();
  if (!rtc.begin()) {
    Serial.println("Couldn't find the RTC module.");
    while (1);
  }
  if (! rtc.isrunning()) {
    Serial.println("The RTC module is not working.");
  } else {
    // Comment out this line after the first run, it is only needed for setting
    // the date and time for the first time.
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }

  // Setup the SD card
  Serial.print("Initializing SD card...");
  if (!SD.begin(SD_PIN)) {
    Serial.println("ERROR: Cannot initialize SD card");
    return;
  }
  Serial.println("SD card is ready.");
  // Now remove the file
  if (SD.remove(LOGFILE)) {
    Serial.println("Cache emptied. File removed.");
  }
```

```
  // Setup the network connection
  Ethernet.begin(mac_addr);

  delay(1000);
  // Test the connection
  Serial.println("Testing connection...");
  if (conn.connect(server_addr, 3306, user, password)) {
    delay(1000);
    Serial.println("Success!");
    conn.close();
  } else {
    Serial.println("ERROR: Connection failed.");
  }
}

void loop() {
  int value;

  // Read the sensor
  value = analogRead(sensorPin);

  // Attempt to connect to the database server
  if (!conn.connected()) {
    if (conn.connect(server_addr, 3306, user, password)) {
      delay(1000);
      dump_cache();  // read any rows stored in the cache file
    }
  }
  // Save the data to the database if connected
  if (conn.connected()) {
    Serial.println("Writing data");
    write_data(value, "NULL");
  } else {
    // Database unreachable, save data to the cache file
    cache_data(value);
  }
  conn.close();
  delay(2000);        // wait for it...
}
```

I want to call your attention to two things in this code. First, notice the last delay() method. Use this method to set the amount of time you want the sketch to wait to take samples. Since we are just testing the sketch, you can keep this short (2 seconds), but you may want that much longer should you implement this code or a derivative for your own data collectors. Second, notice I have added many informational messages. Some would suggest I've added too many, and indeed if you took them out, it would save some space, but I put them there so that I could see how the sketch was running (more precisely, when the data is cached, read from cache, and written to the database). Let's see that it action.

Test the Sketch

Now let's get the code compiled and uploaded to the Arduino Mega. One thing you will notice when you compile the code is that it consumes more than 36KB of program storage. While the compiler may show something similar to the following and indicate you have plenty of space available, if you were to attempt to compile this for a smaller Arduino, you will get an error stating the program is too large.

```
Sketch uses 36,370 bytes (14%) of program storage space. Maximum is 253,952 bytes.
Global variables use 2,219 bytes (27%) of dynamic memory, leaving 5,973 bytes for local
variables. Maximum is 8,192 bytes.
```

Once the code is compiling properly and you have set up the sample database, go ahead and run it for some time. After a while, you can simulate a database failure by disconnecting the Ethernet cable. Let's see what happens.

If you open the serial monitor, you will see the output of the information messages I sprinkled throughout the code. Listing 8-8 shows an excerpt of a run. Notice that the sample values are the same. Recall I set the delay to only two seconds, so there will not be much variance in the sample values. In a real-world implementation, you would probably want to check the soil moisture once per hour or even once every couple of hours for plants in a controlled climate.

Listing 8-8. Debug Statements from Fault-Tolerant Data Collector

```
Connected to server version 5.7.8-rc-log
Writing data
INSERT INTO plant_moisture.plants VALUES (NULL,456,NULL)
Disconnected.
Connected to server version 5.7.8-rc-log
Writing data
INSERT INTO plant_moisture.plants VALUES (NULL,456,NULL)
Disconnected.
Data cached to log file.
Data cached to log file.
Data cached to log file.
Connected to server version 5.7.8-rc-log
INSERT INTO plant_moisture.plants VALUES (NULL,456,'2015-11-27 15:9:8')
INSERT INTO plant_moisture.plants VALUES (NULL,457,'2015-11-27 15:9:21')
INSERT INTO plant_moisture.plants VALUES (NULL,455,'2015-11-27 15:9:34')
Cache emptied. File removed.
Writing data
INSERT INTO plant_moisture.plants VALUES (NULL,456,NULL)
Disconnected.
```

Notice that when I disconnected the Ethernet cable, the code saved the data to the cache file. But later when I reconnected the cable, the data was read from the cache and inserted in the database. Notice those three statements again. Do you see that the date and time values were preserved? Recall this data was saved to the file. But notice the other INSERT statements use NULL to signal the database to use the current date and time of when the data was saved. That's pretty neat, yes?

Check the Requirements

How did we do with the requirements? I like to check the requirements as I've written them against the code once it is finished and working. Actually, I check the requirements at every stage, but checking them once the sketch has run for a bit will help you examine the data to ensure data was saved correctly. Let's go through each requirement (noted by the numbers inline).

From the test, we see all the data was saved (1). We do a test and capture the loss of communication to the database server (2). Data is cached to a file on the SD card when the connection is lost (3) (4). When reconnected, any cached data is written to the database (5), and the cached data's date and time of sample collection are preserved (6).

There you have it—your first fault-tolerant data collector with recovery. While I used Ethernet in this example, which is generally more reliable, you can modify this code to use other forms of communication and thus protect your data collection from loss from communication mechanism failure.

Using techniques like this along with MySQL replication for redundancy, you can achieve an impressive form of high availability for your IOT solutions. I encourage you to continue to explore code like this for other nodes hosted on platforms such as Raspberry Pi, Edison, and other single computer boards. What you will find is those boards have sufficient memory to build even more sophisticated fault-tolerant mechanisms.

Summary

High availability is all about reliability. The more reliable a solution is, the greater the level of high availability. The concepts of high availability therefore are not difficult but can be a little challenging to implement. Fortunately, MySQL provides a number of high availability features; chief among them is MySQL replication.

In this chapter, you learned more about MySQL replication including some helpful tips and techniques for setting up and using replication in your IOT solutions. You also saw some examples of high availability concepts including a simple round-robin read scale-out solution for Python, hardware for building a redundant data collector with failover, and how to build in fault tolerance for data collectors so that you don't lose data should the communication pathway from data collector to database fail.

As you have learned, it is possible not only to achieve some level of high availability in IOT solutions; you can do so from the lowest level of the IOT network to the highest. From hardware to database to software application, all can be made to be more reliable and ultimately reach high availability.

Index