CHAPTER 7

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.