# MySQL for the Internet of Things

A MySQL Whitepaper

Charles Bell, PhD

# Table of Contents

# Introduction

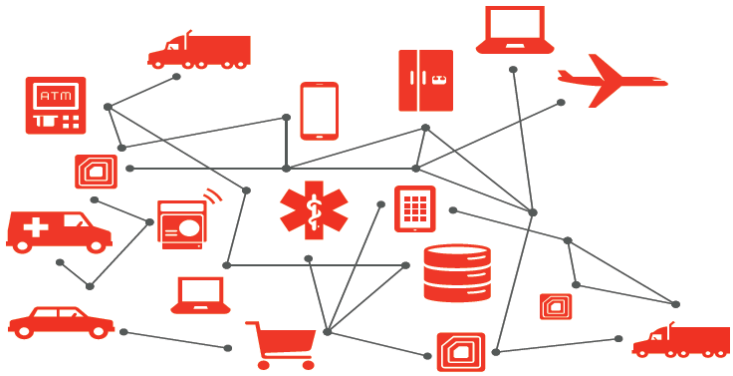
The Internet of Things (IoT) is poised to change how we interact with and perceive the world around us, and the possibilities are nearly boundless. As more and more connected devices generate data, we will need to solve the problem of how to collect, store and make sense of IoT data by leveraging the power of database systems.

In this whitepaper, we explore what the IoT is, how IoT data is collected and stored, as well as how to leverage MySQL in your IoT solution.

# 1. 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 so that we can learn more about the world around us.

Thus, the IoT is all about connecting things to each other and providing interfaces for humans to observe something, be that through sensor data, published facts, derived data, etc. – all of this is available to anyone via the Internet from anywhere. The IoT therefore helps us gain knowledge of the world around us.




However, the IoT is not just about connecting something to the Internet. There must be some benefit to someone or yourself – there must be some knowledge gained or shared. That is, simply connecting a device to the Internet doesn't make it an IoT solution. At best, you can call it "Internet enabled". There are many applications you could make into an IoT solution, but do they all make sense?

For IoT solutions that are meaningful, designing and collecting data from IoT devices presents two significant problems: big data and addressing. However, big data and addressing are largely a product of applying a concept universally without much forethought. As we will see, we can apply database best practices and goals to IoT data.

# 2. The IoT and Big Data

Big data is simply a case where you have more data than any one system or even systems of systems can process efficiently. That is, you have far too much data to process in any reasonable timeframe (but there are ways to do so).

You may be surprised to learn that some feel big data is a myth or a consequence of trying to store everything in the "cloud" – it just fills up beyond any conceivable method to process. However, big data isn't just "about the cloud" nor is it a myth. Consider for a moment what would happen if every device all over the world were to generate 100 data elements every second. That could easily turn into a huge amount of data in only a few hours.

Is that reasonable and where do we put this data? Is it even relevant or is there any correlation with other data? Furthermore: what computing resources do we need to store and process the data? Wouldn't all of the data be just a big clutter of data?

There is one way to reduce the clutter: we can apply carefully crafted data transformation techniques to pare down the data and make it more meaningful ("the less is more" principle).

For example, if those devices all over the world were recording an observation that is not prone to major changes in a short time period – say ambient temperature – is there any need to store 100 samples every second? It may be more reasonable to store 1 sample every few minutes or even a few samples every hour. Thus, by selecting a more reasonable sampling rate, we've reduced the "big data" problem significantly. We will see more examples of transforming the data in later sections.

# 3. The IoT and Addressing

Addressing is a similar problem. Some believe the IoT means every device everywhere is connected to the Internet and all have a public IP address that you can connect to. That's not practical. Why would you do that and how would you find these devices if you were looking for them?

Placing a unique number on every rock in the desert may indeed ensure every rock is addressable, but does it make them locatable? Do the IP addresses have anything to do with geography? Generally speaking, no. Thus, brute force searches for a given device (or IP) in the sea of IPv6 addresses could take years even with the best and fastest search engines.

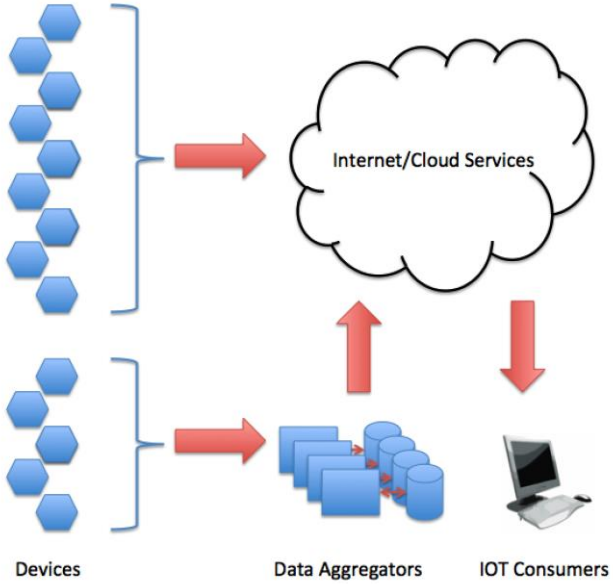
What we need then is a brokerage system where forward-facing systems (devices perhaps) can advertise or publish their capabilities. This frees the IoT devices "behind" the broker to use any networking protocol they want. Cloud systems and vendors offer a variety of solutions in this area. Thus, we have effectively reduced the addressing problem and made searching faster.

# 4. What is IoT Data?

IoT data is, like any other application, the most valuable aspect of the system – or at least the most valuable by-product of the system. The data must be reliable and readily accessible and depending on the industry or use, may have differing quality needs. The medical industry would most definitely require a higher level of quality for precise analysis than say the advertising industry, which can operate from statistical relevance.

However, IoT data isn't and should not be considered simply anything that comes out of a sensor or human-generated observation. The data needs to have context in order to be relevant. Again, if you were recording temperature readings in your home, those data points without context such as the room, time of day, outside weather observations, whether the room was occupied (and by whom), or even the settings of your environmental systems may not be very useful. But if you added that context, you may be able to analyze the data to help reduce your energy costs by programming the environmental system to match trends of temperature changes in your home throughout the day taking into account the effects of the additional data.

Thus, with context, IoT data can help us learn about world around us. Let's take a look at a hypothetical IoT solution from the perspective of how the IoT devices connect to the Internet.




Here we see some devices may not connect to the Internet at all whereas some may connect to one or more data aggregators – devices or systems that aggregate or augment the data in some way perhaps even storing it.

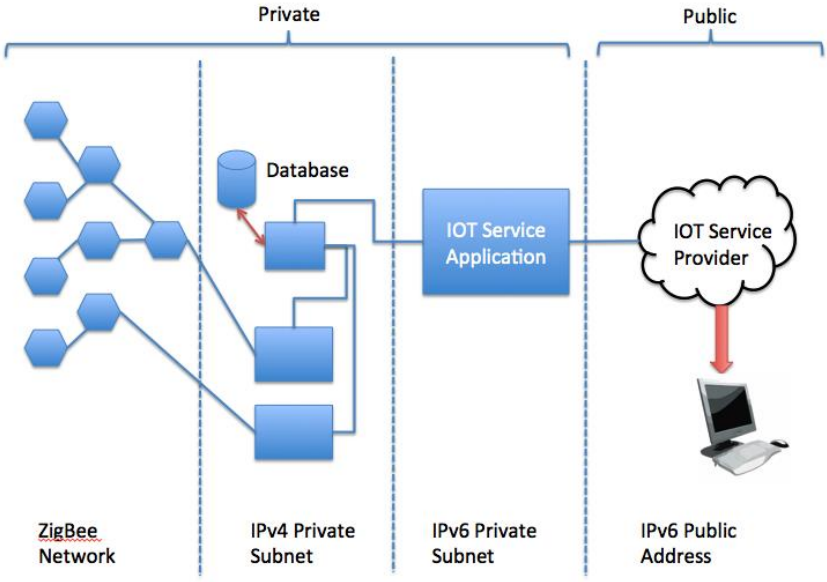
Those devices connecting via the Internet will need IP addresses while those that do not are free to use some other form of networking protocol.

But the important consideration here is the data aggregators or similar intermediate nodes. These can be used to pare down the data, add context (additional knowledge), and even "clean" the data by correcting errors such as scale or sensor variations or limitations (depending on the technology, sensors may need to be interpreted).

# 5. Designing IoT Systems

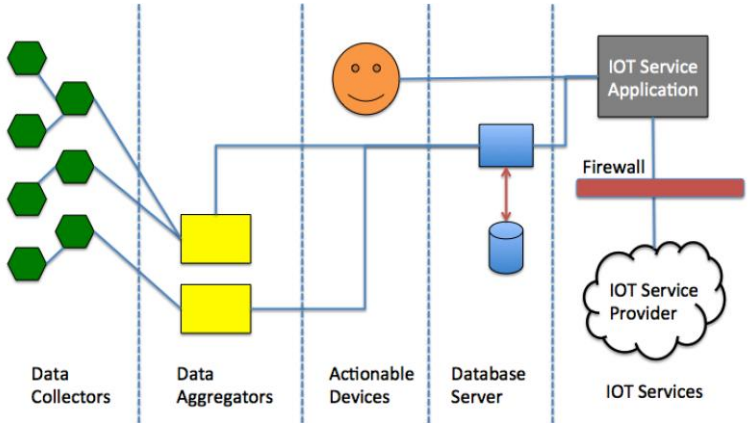
Using this concept, we can form a network of our IoT solution with a private zone (those devices behind the IoT service application) and public zone (the IoT service application and IoT service provider and/or IoT services).

In the private zone, we can employ whatever networking protocols we want from wireless such as ZigBee to private IP addressing with WiFi or Bluetooth. We only need public IP addresses in the devices in the public zone.




IoT networks can be constructed using one or more of several types of IoT devices such as the following.

- Data collector: A sensor, IOT device, and so on that produce data from some event or observation.
- Data aggregator: A node that receives information from one or more data collectors. Aggregates and augments 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, etc.
- 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.




Here we see how each of the types fits into a hypothetical IoT network of devices. On the left we see the data collectors followed by the data aggregators, actionable devices, the database server, and finally the public zone of devices including the IoT service application and IoT services. This layered model helps visualize how data moves from observation to knowledge and is distributed to the world.

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.

No discussion about IoT solutions would be complete without addressing security. Security in IoT solutions should be the same techniques and practices as any other application. In fact, you should go a bit further and decide which parts need to be made public and which to remain private. If nothing else, IoT developers need to be more vigilant about security.

A now famous example is the hacking of certain automotive infotainment systems. While the manufacturer has since closed the security loopholes, the system was vulnerable to attacks and exploitation via the Internet with dramatic consequences such as loss of steering control.

Thus, you should take measures to secure your IoT solution from security vulnerabilities both real and imagined.

# 6. Storing IoT Data with MySQL

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.

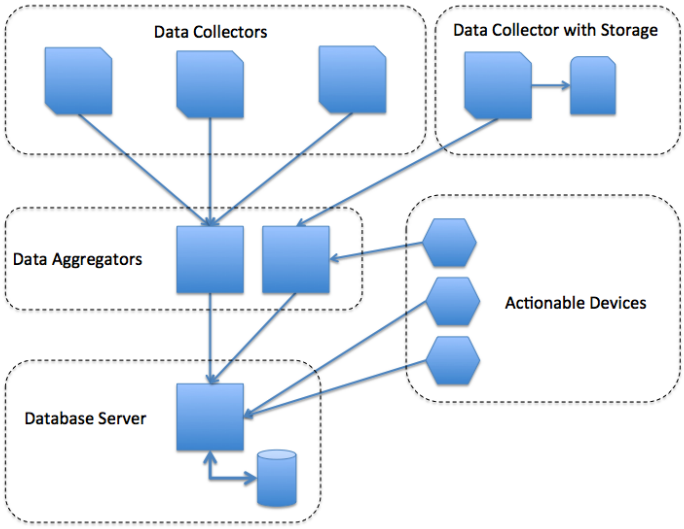
Not only do you have to figure out how to scale the communication among your sensors, data collectors, and data-hosting services or database servers, you also have to deal with an explosion of data. That is, capturing data from a dozen sensors is relatively easy and may not require much in the way of 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.

With all this talk about the data and how important it is to make it meaningful and secure, how do we go about storing it?

You could store the data locally on each node and publish the data in the cloud. Or you could not store anything and put all of the data in the cloud. But either way, it means adding a data access layer (or service) that may not be a good fit for the data (or vice-versa).

A better method involves distributing the data among your IoT network nodes. For example, you could store the raw data on data aggregators and the augmented or aggregated data on one or more database servers. Not only does this permit more powerful transformation (via the data aggregators), but it also allows you some redundancy and the ability to publish better data than simply throwing out whatever the IoT devices collect.

Let's look at this concept a bit deeper. Suppose we had a distributed IoT data model using the node types described previously. In this case, we can add local storage to the data collectors and data aggregators. This could be in the form of files, memory, or in the case of low-cost computing boards, a MySQL server.




If we then transport all of the transformed data to a local database (an embedded MySQL server), we have gained more knowledge with better data (data with context for example), while retaining the raw data on the nodes in the network. Thus, it is possible to recover the original, raw data.

Does storing IoT data in MySQL mean we have to change how we collect data or learn new features of MySQL? Absolutely not. You can leverage all of your existing MySQL knowledge and apply it directly to your IoT data. That is, you can write your queries in SQL, your data is more secure than storing in files or memory, you can perform post-data collection processing, and more. Best of all, MySQL is a very popular choice as an embedded database, used by 8 of the 10 largest software vendors in the world to power their products.

The following are just a few of the market-leading ISVs and OEMs that use MySQL as an embedded database.


You may be wondering how you connect your microcontroller to MySQL. Despite the rather low processing power available in the smaller Arduino boards, there exists a database connector that allows you to write data to a MySQL server directly from the Arduino. It's called Connector/Arduino and is published in the Arduino Library Manager and available for installation from inside the IDE.

For low-cost computing boards, you can use any of the existing database connectors found on dev.mysql.com such as Connector/Python or Connector/J.

# 7. Data Transformations for Better IoT Data

What is this data transformation and how can it make my IoT data more manageable and produce better knowledge? First, we must make sense of the data.

Making sense of the data is about understanding the knowledge that the data will give us. We arrive at what needs to be annotated or aggregated by asking ourselves questions like the following.

- What is being observed?
- What do we want to learn?
- Is there another way to make the observation?
- How often do you need to make the observation?
- What type of data does the sensor produce?
- Are there interpretations needed for the observation data?
- What level of accuracy do you need?
- What is the lifetime of the data?

The answers to these questions (and others like them) will help us understand how the data is used and furthermore how useful the data could be if we added more context. For example, if there are multiple interpretations of the data, we should store them.

Consider the raw IoT such as data from sensors. Often times, the data is in some scale perhaps analog voltage, a number from -256 to +256, etc. that needs to be deciphered or translated into a scale or measurement that makes sense to humans. It is meaningless to say, “it’s -2.3 volts out there so bundle up!”

Thus, we often have to make sense of the data before we can garner any meaning from it. The trick then, is converting it in such a way as to make it human-decipherable without losing data. But this sometimes means adding more data. We do so through annotation and aggregation.

Annotation is the process of adding things to the data. However, we don’t add any random thing. We add things that add meaning such as storing the source of the observation, the date and time of its observation, transforming it into another scale, or calculations on the data (such as scaling or error correction), and finally interpreting the data (such as converting from analog values to floating point numbers in a scale).

For example, recording date and time is really easy with MySQL. That is, we often need to store the date and time when an event or series of events are observed. We want to save the date and time when the sensor was read.

Saving a date and time in a table 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. The following shows an example table schema that includes a single date and time 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.');
```

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;
```

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 value and in the case of a timestamp column, the server will use the current date and time of when the row was inserted. Thus, not only does the timestamp column mean the database does the work for you, it also means you don't have to use any additional coding or hardware in your data collectors or sensor nodes to work with date and time. The database can keep track of when the data was added or changed.

Aggregation on the other hand is taking data from multiple sensors or data collection nodes and performing calculations such as average, min, max, and other statistical operations. We may also need to aggregate data using different rates. For example, soil moisture changes much more slowly (instantaneous flooding by a watering can notwithstanding) than temperature in which a plant is placed. Thus, these data have differing storage requirements even though they are often used together in analysis, which means we have to design our database a bit differently to accommodate aggregate functions on the data.

For example, suppose 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.

Consider the following code excerpt from a Python script. 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.

```
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.

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.

The following shows the use of 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).

```
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.

Data transformation is not just about taking one data type and changing it to another. As you saw in this section, 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.

# 8. MySQL High Availability IoT Solutions

While high availability is often defined slightly differently depending on how it is applied (or what the overall goals are), for IoT solutions, we can consider high availability synonymous with reliability. In other words, how accessible the data is during any given time. That is, how reliable the system is in providing the data for consumers.

But why should we care? Consider the loss of data in the confines of IoT is a loss of knowledge. Depending on the application area, this could be huge.

Furthermore, we should expect our users to be less happy with a system that cannot recover from minor (or even moderate) outages. Too much of this and customers may be less inclined to trust it.

To understand the value of high availability, let's consider what could go wrong such as:

- What do you do if a sensor node goes offline?
- What do you do if a data aggregator goes offline?
- What about the database server?

The answers to these questions and the actions you would take to prevent or recover from depend on how much you value your data. If the data is very important, you will want to take steps to make the solution more robust.

There are three primary tasks you can undertake to achieve high availability in your IoT solutions.

- Eliminate single points of failure – use fewer components allowing for faster replacement
- Add recovery through redundancy – add redundant systems/nodes
- Implement fault tolerance – detect and react to failures leveraging redundancy

There are also a number of goals we should consider for high availability and IoT. We can leverage backup and restore tools to recover the data from outage or loss, we can use redundancy in our databases (replication) to allow switchover or failover, we can use scaling to improve performance, and finally, we can use fault tolerance to automatically recover from failures.

| 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 |

Fortunately, there are a number of MySQL solutions you can leverage for these high availability goals.

MySQL Enterprise Edition offers a wide range of supported high availability solutions as well as advanced monitoring tools and support services, that help you achieve high availability (in any solution).

The monitoring tools in MySQL Enterprise Edition are very powerful features. The ability to tune queries alone can help improve performance. Altogether, the MySQL Enterprise Monitor with Query Analyzer can even help improve your DevOps (that's what it was built for!).

For backup and recovery goals, we can use MySQL Enterprise Backup (MEB), the client tools that ship with the server (mysqldump, mysqlpump), and MySQL Utilities to backup and recover our data.

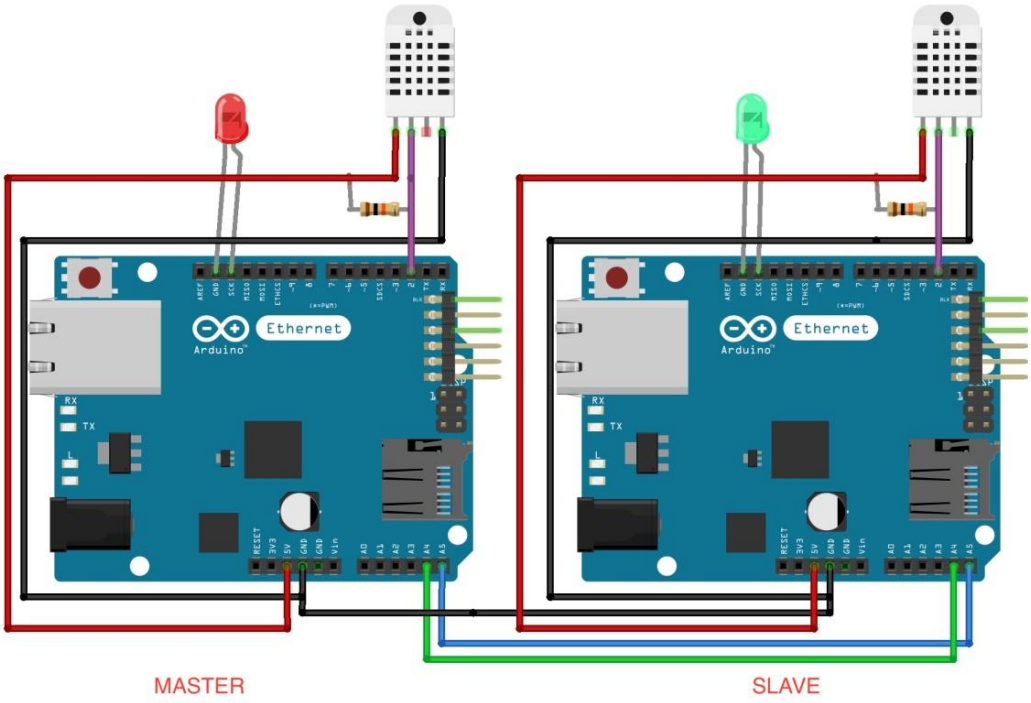
For redundancy goals, we can use MySQL replication for the database nodes, redundant hardware, backup power sources (such as battery+solar+mains for remote data collectors) or simply log data on data aggregators for fast recovery from more severe outages.

For scaling goals, we can use MySQL replication including the Global Transaction Identifier (GTIDs) feature, or MySQL Cluster for in-memory, high availability needs.

For fault tolerance goals, we can leverage several MySQL features already mentioned including MySQL utilities for single master failover, MySQL Fabric for farm-level fault tolerance, and MySQL cluster for even more robust recovery.

Fault tolerant goals for the hardware include techniques for failover to auxiliary hardware nodes, using multiple sensors, and even multiple data aggregator nodes in a master/slave arrangement.

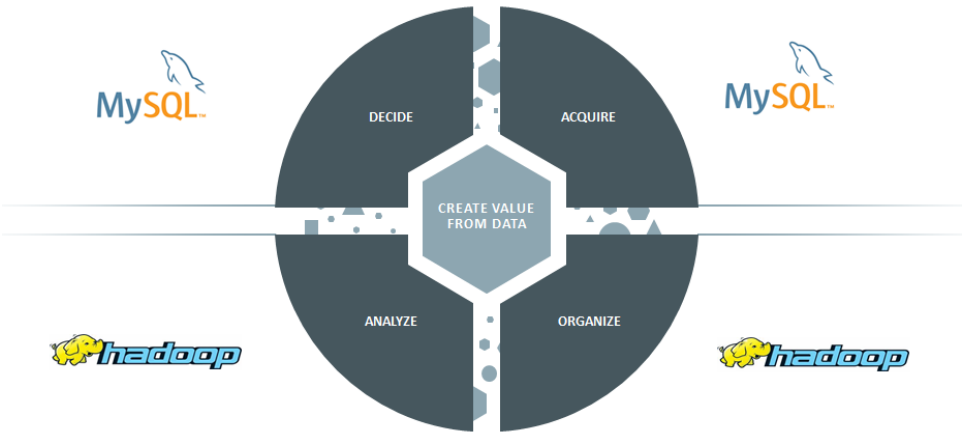
Here's an example of IoT hardware fault tolerance: a redundant data collector. Here we see two microcontrollers wired in series each with its own sensor(s). One acts as the master sending a heartbeat to the slave. The slave monitors the heartbeat and should it stop (notice the connection is hardwired, not relying on networking), the slave takes over. At this point, the slave could notify or log the event so that you can later repair and place the master back in service.



# 9. Data Processing & Analysis

Once collected, data needs to be processed and analyzed in order to generate new insights.

With the exponential growth in data volumes and data types generated partly by the IoT, it is important to consider the complete lifecycle of data, enabling the right technology to be aligned with the right stage of the lifecycle. We will consider MySQL in the Big Data Lifecycle as well as the technologies and tools at your disposal at each stage.




**The Data Lifecycle**

As illustrated in the above figure, the lifecycle can be distilled into four stages:

**Acquire:** Data is captured at source, typically as part of ongoing operational processes. Examples include log files from a web server, user profiles and orders stored within a relational database supporting a web service or IoT applications as discussed in this white paper.

**Organize:** Data is transferred from the various operational systems and consolidated into a big data platform, i.e. Hadoop / HDFS (Hadoop Distributed File System). Data can be transferred in batches from MySQL tables to Hadoop using Apache Sqoop.

**Analyze:** Multi-structured data ingested from multiple sources is consolidated and processed within the Hadoop platform. Data stored in Hadoop is processed either in batches by Map/Reduce jobs or interactively with technologies such as the Apache Drill or Cloudera Impala initiatives. Data can also be processed in Apache Spark. Hadoop may also perform pre-processing of data before being loaded into data warehouse systems, such as the Oracle Exadata Database Machine.

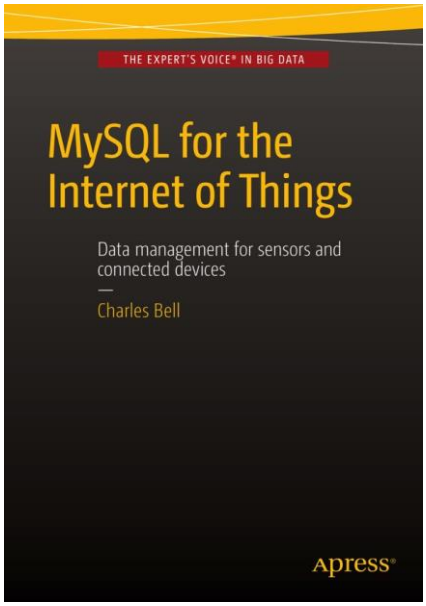
**Decide:** The results of the analysis are loaded back to MySQL via Apache Sqoop where they power real-time operational processes or provide analytics for BI tools. For example, the data maybe loaded back into the operational MySQL database supporting a web site, enabling recommendations to be made to buyers; into reporting MySQL databases used to populate the dashboards of Business Intelligence tools, or into the Oracle Exalytics In-Memory machine.

For more information, please download our “Unlocking New Big Data Insights with MySQL” white paper at: http://www.mysql.com/why-mysql/white-papers/mysql-and-hadoop-guide-to-big-data-integration/

# Conclusion

In summary, we've explored how carefully constructed IoT solutions can leverage data transformation to reduce the burden of Big Data; we've discussed how we can employ several technologies for working with IoT data, and reviewed how to leverage Oracle's MySQL solutions within your architecture, both for data collection and analysis.

If you would like to know more about how to leverage MySQL in your IoT solutions including more details about transforming IoT data as well as practical examples and sample code for transforming IoT data, see the book entitled, *MySQL for the Internet of Things* (Apress 2015) by Charles Bell (http://www.apress.com/9781484212943?gtmf=c).




# Additional Resources

- MySQL Resources, including White Papers, Demos, Forums and more
  http://www.mysql.com/why-mysql/

- MySQL Webinars:
  - Live: http://www.mysql.com/news-and-events/web-seminars/index.html
  - On Demand: http://www.mysql.com/news-and-events/on-demand-webinars/

- "MySQL for the Internet of Things" (Apress 2015), Charles Bell
  http://www.apress.com/9781484212943?gtmf=c

- MySQL Products
  http://www.mysql.com/products/