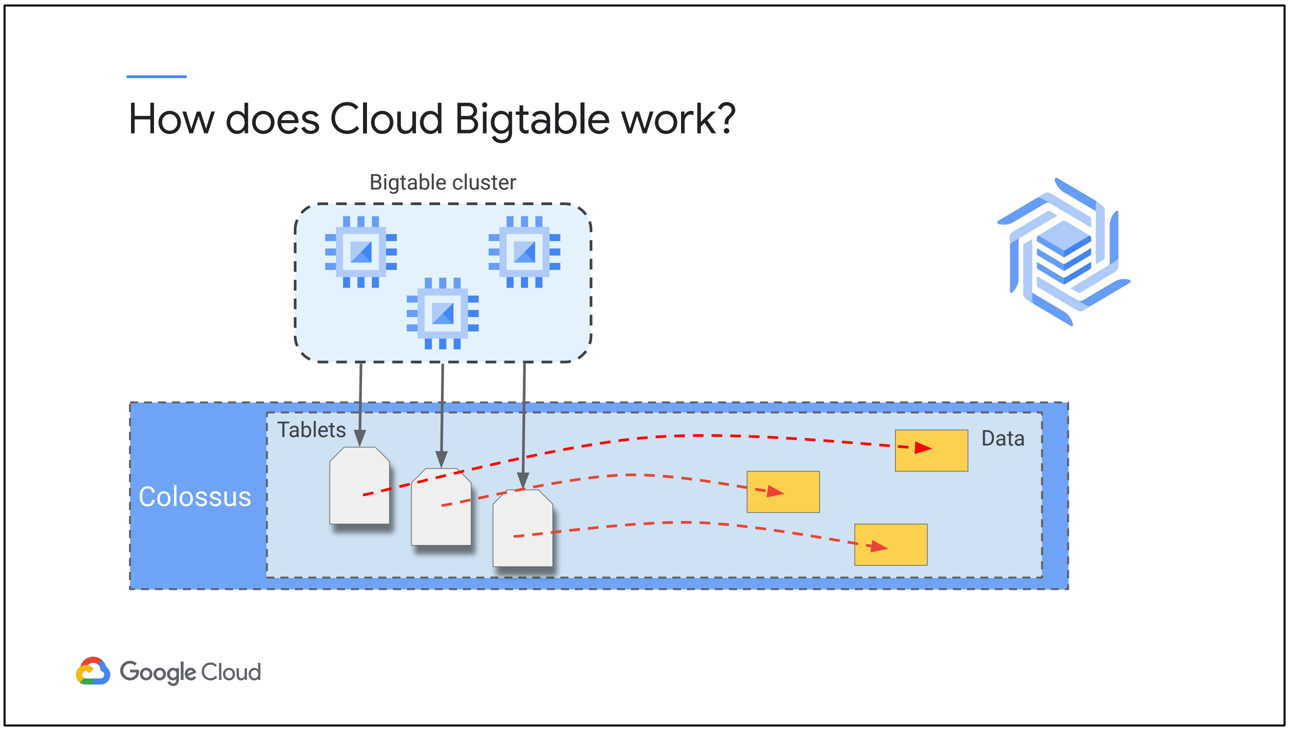
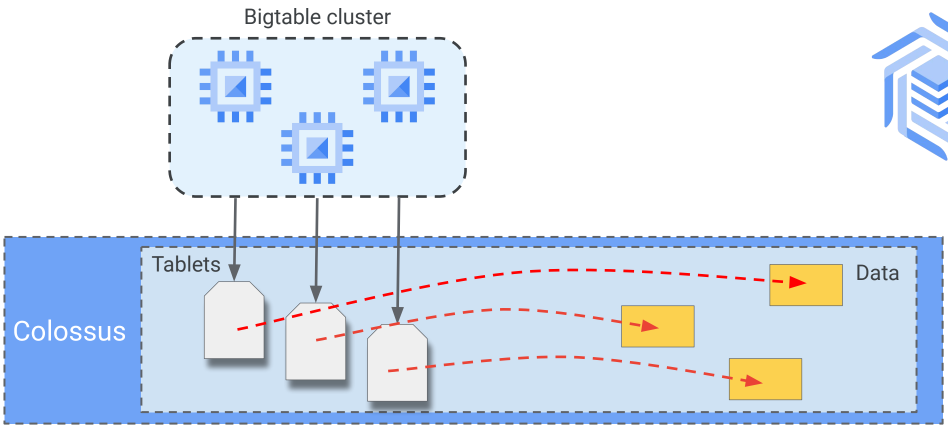


How does Cloud Bigtable work?



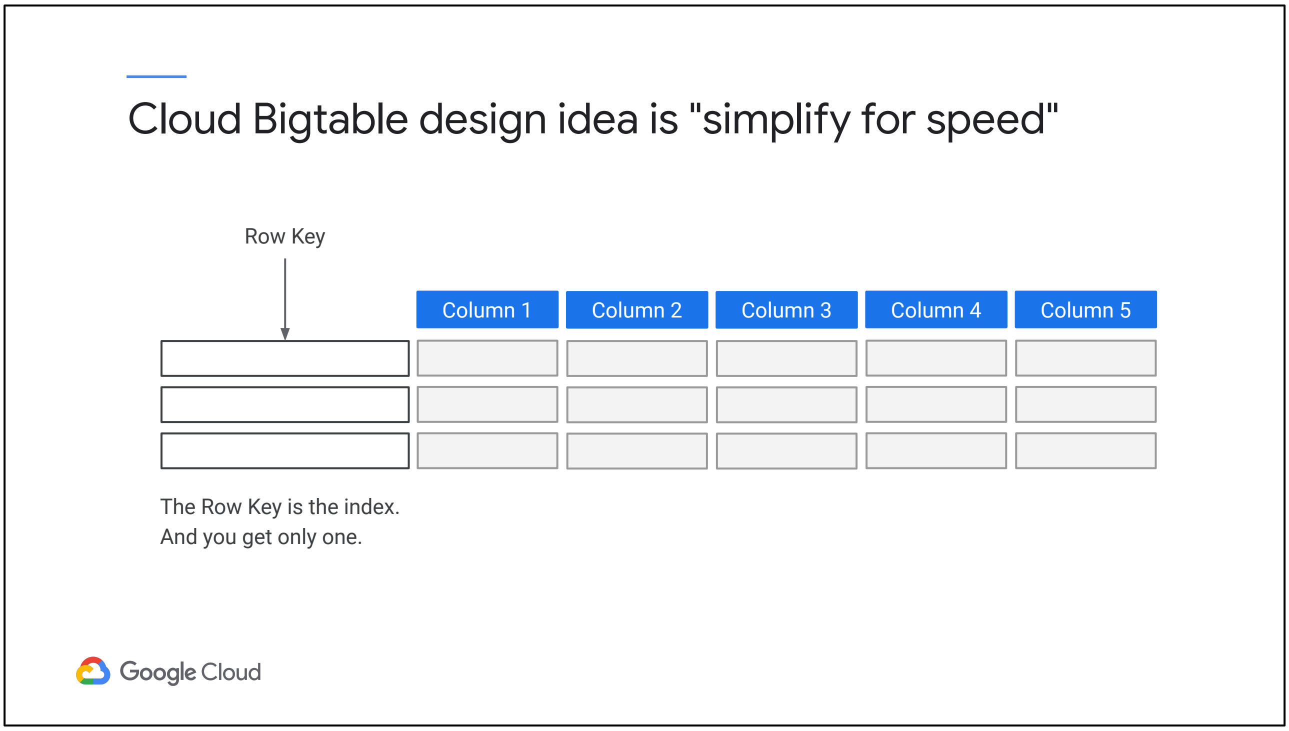
Google Cloud

Cloud Bigtable stores data in a file system called Colossus. Colossus also contains data structures called Tablets that are used to identify and manage the data. And metadata about the Tablets is what is stored on the VMs in the Bigtable cluster itself.

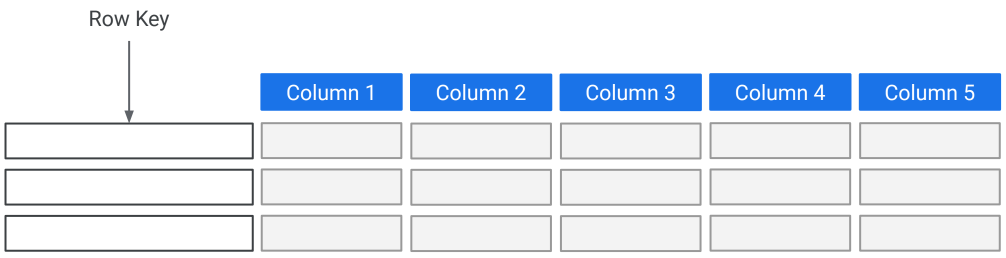
This design provides amazing qualities to Cloud Bigtable. It has three levels of operation. It can manipulate the actual data. It can manipulate the Tablets that point to and describe the data. Or it can manipulate the metadata that points to the Tablets. Rebalancing tablets from one node to another is very fast, because only the pointers are updated.

Cloud Bigtable is a learning system. It detects "hot spots" where a lot of activity is going through a single Tablet and splits the Tablet in two. It can also rebalance the processing by moving the pointer to a Tablet to a different VM in the cluster. So its best use case is with big data -- above 300 GB -- and very fast access but constant use over a longer period of time. This gives Cloud Bigtable a chance to learn about the traffic pattern and rebalance the Tablets and the processing.

When a node is lost in the cluster, no data is lost. And recovery is fast because only the metadata needs to be copied to the replacement node.
Colossus provides better durability than the default 3 replicas provided by HDFS.



Cloud Bigtable design idea is "simplify for speed"



The Row Key is the index.
And you get only one.

Google Cloud

Cloud Bigtable stores data in tables. And to begin with, it is just a table with rows and columns. However, unlike other table-based data systems like spreadsheets and SQL databases, Cloud Bigtable has only one index.

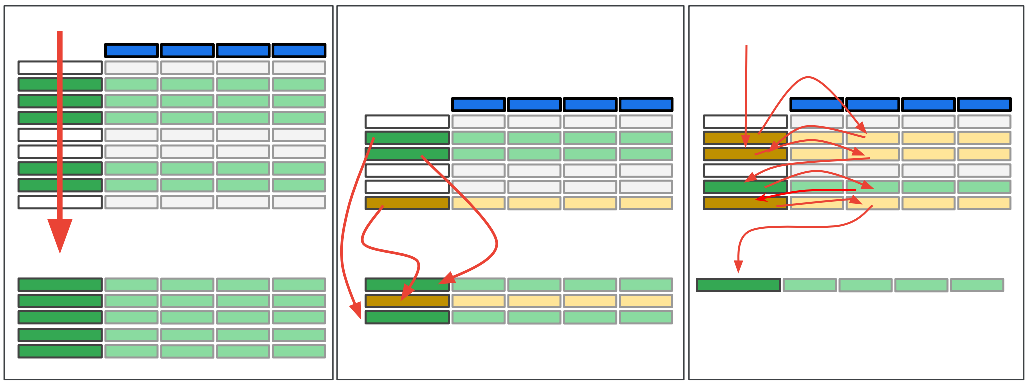
That index is called the Row Key. There are no alternate indexes or secondary indexes. And when data is entered, it is organized lexicographically by the Row Key.

The design principle of Cloud Bigtable is speed through simplification. If you take a traditional table, and simplify the controls and operations you allow yourself to perform on it then you can optimize for those specific tasks.

It is the same idea behind RISC (Reduced Instruction Set Computing). Simplify the operations. And when you don't have to account for variations, you can make those that remain very fast.

In Cloud Bigtable, the first thing we must abandon in our design is SQL. This is a standard of all the operations a database can perform. And to speed things up we will drop most of them and build up from a minimal set of operations. That is why Cloud Bigtable is called a NoSQL database.

But speed depends on your data and Row Key



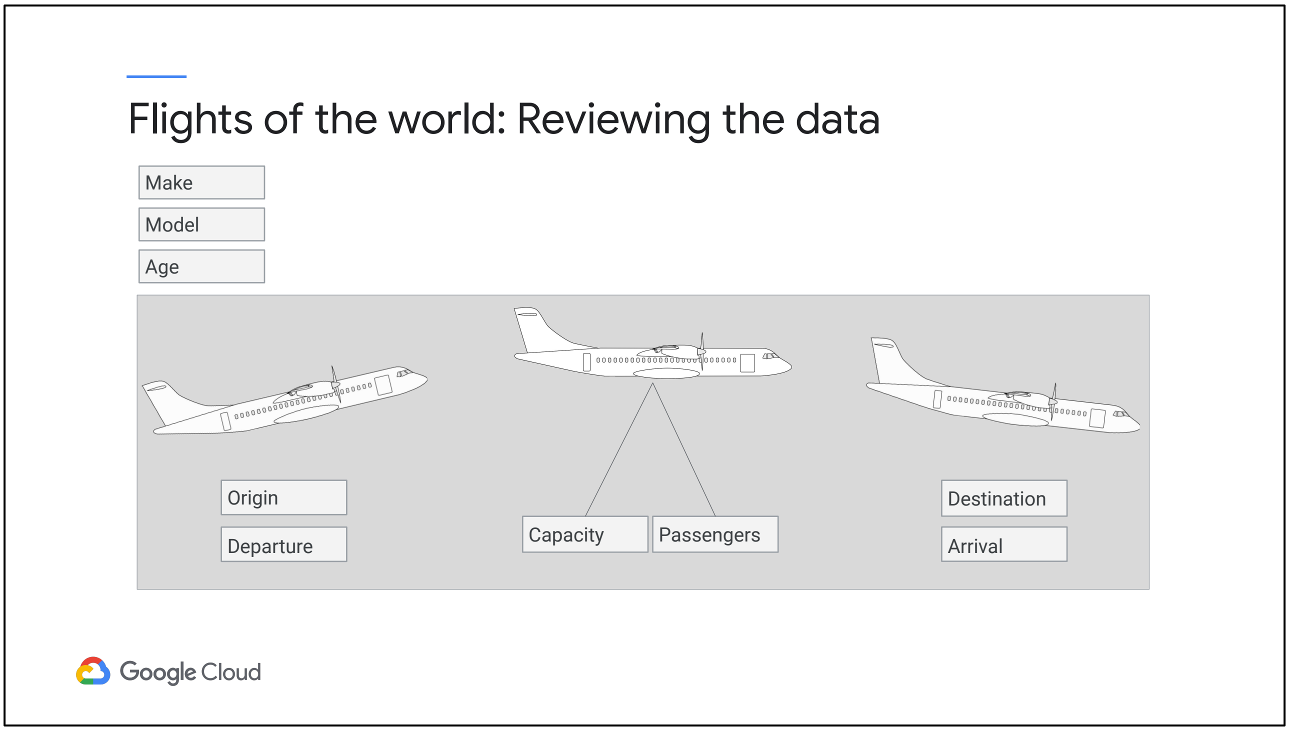
The green items are the results you want to produce from the query. In the best case you are going to scan the Row Key one time, from the top-down. And you will find all the data you want to retrieve in adjacent and contiguous rows. You might have to skip some rows. But the query takes a single scan through the index from top-down to collect the result set.

The second instance is sorting. You are still only looking at the Row Key. In this case the yellow line contains data that you want, but it is out of order. You can collect the data in a single scan, but the solution set will be disorderly. So you have to take the extra step of sorting the intermediate results to get the final results. Now think about this. What does the additional sorting operation do to timing? It introduces a couple of variables. If the solution set is only a few rows, then the sorting operation will be quick. But if the solution set is huge, the sorting will take more time. The size of the solution set becomes a factor in timing. The orderliness of the original data is another factor. If most of the rows are already in order, there will be less manipulation required than if there are many rows out of order. The orderliness of the original data becomes a factor in timing. So introducing sorting means that the time it takes to produce the result is much more variable than scanning.

The third instance is searching. In this case, one of the columns contains critical data. You can't tell whether a row is a member of the solution set or not without examining the data contained in the critical column. The Row Key is no longer sufficient. So now you are bouncing back and forth between Row Key and column contents. There are many approaches to searching. You could divide it up into multiple steps, one scan through the Row Keys and subsequent scans through the columns, and then perhaps a final sort to get the data in the order you want. And it gets much more complicated if there are multiple columns containing critical information. And it gets more complicated if the conditions of solution set membership involve logic such as a value in one column AND a value in another column, or a value in one column OR a value in another column. However, any algorithm or strategy you use to produce the result is going to be slower and more variable than scanning or sorting.

What is the lesson from this exploration? That to get the best performance with the design of the Cloud Bigtable service, you need to get your data in order first, if possible, and you need to select or construct a Row Key that minimizes sorting and searching and turns your most common queries into scans.

Not all data and not all queries are good use cases for the efficiency that the Cloud Bigtable service offers. But when it is a good match, Cloud Bigtable is so consistently fast that it is magical.



Flights of the world: Reviewing the data

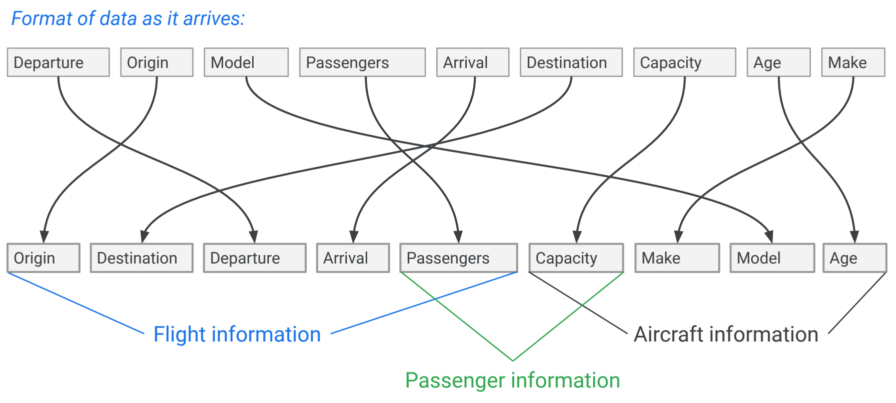
Each entry records the occurrence of one flight.

The data include city of origin and the date and time of departure, and destination city and date and time of arrival.

Each airplane has a maximum capacity, and related to this is the number of passengers that were actually aboard each flight.

Finally, there is information about the aircraft itself, including the manufacturer, called the make, the model number, and the current age of the aircraft at the time of the flight.

Flights of the world: Transforming the data

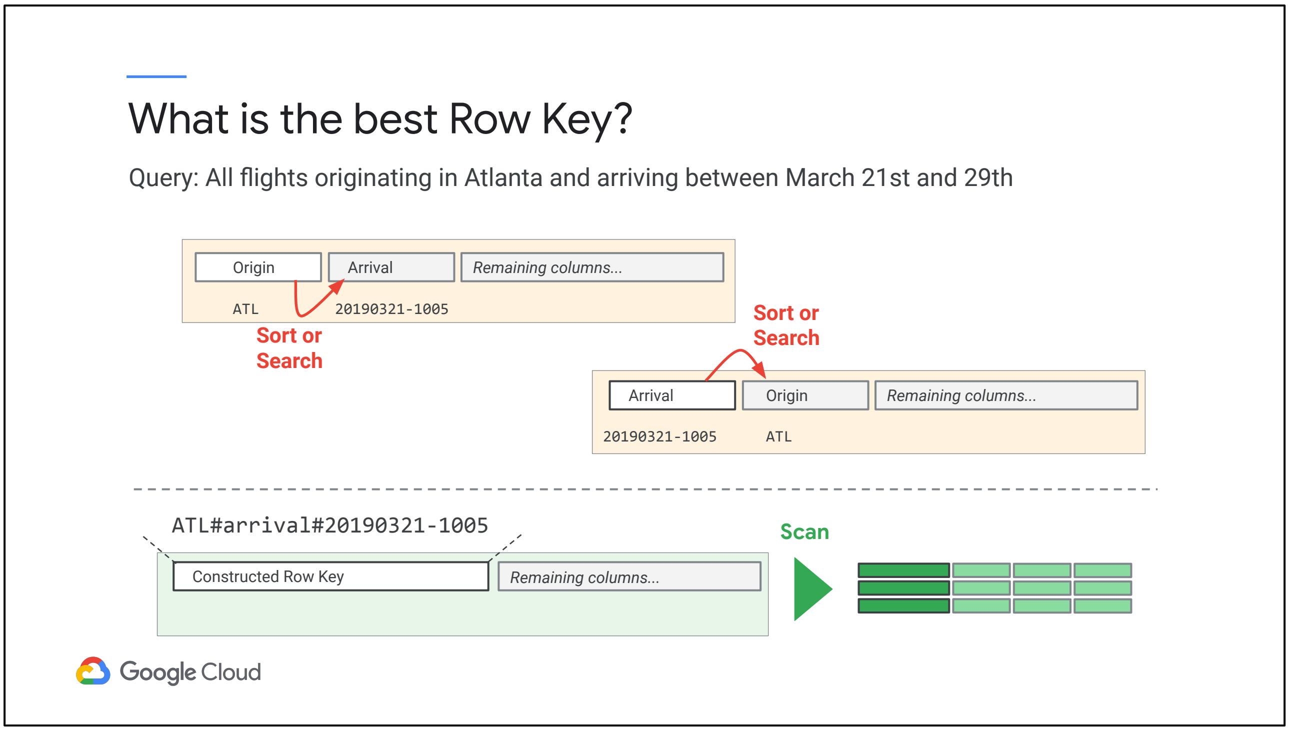


Google Cloud

Often the data arrives in a pre-selected order that might not be optimal for the purposes of the application. In this case, transforming the data before storing it makes sense.

In the example, the fields are sorted into Flight Information and Aircraft Information.

The number of passengers on a flight and the capacity of the flight to hold passengers were moved to the end, and beginning of the groups so that passenger-related information would be adjacent.



What is the best Row Key?

Query: All flights originating in Atlanta and arriving between March 21st and 29th

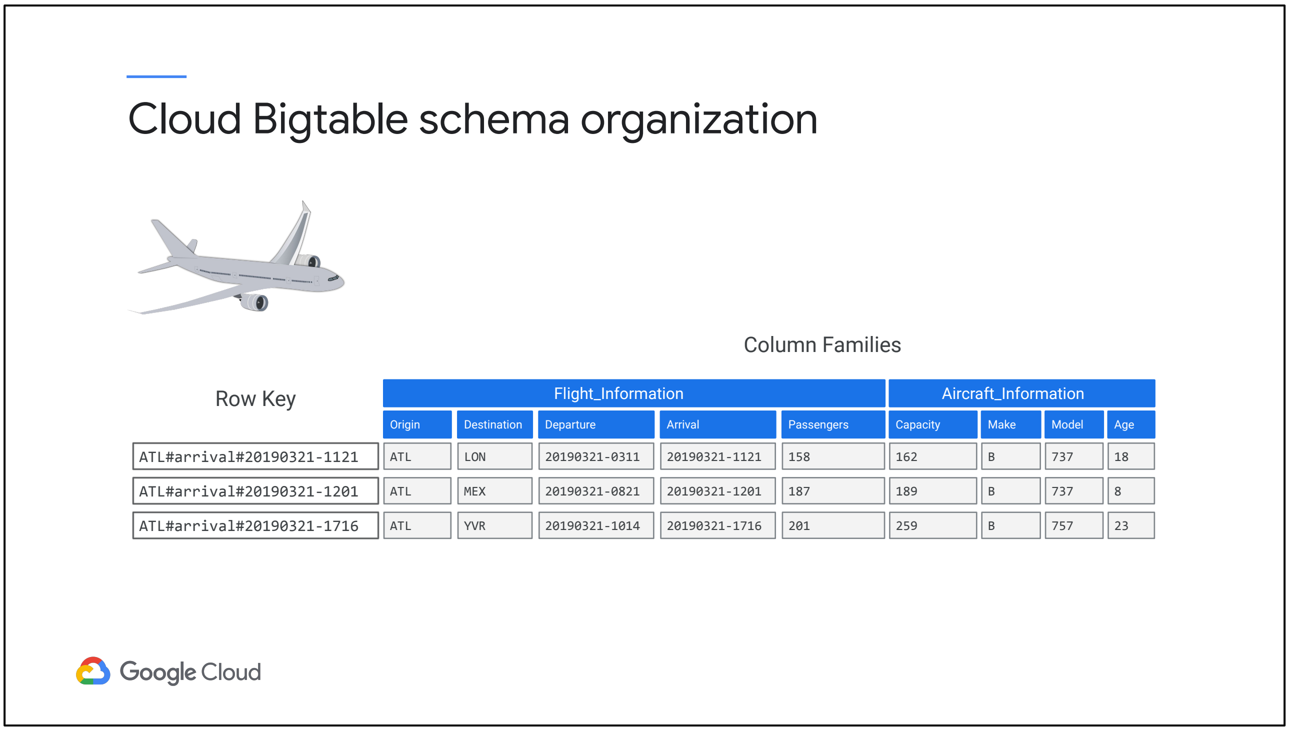
In this example, the Row Key will be defined for the most common use case. The Query is to find all flights originating from the Atlanta airport and arriving between March 21st and 29th. The airport where the flight originates is in the Origin field. And the date when the aircraft landed is listed in the Arrival field.

If you use Origin as the Row Key, you will be able to pull out all flights from Atlanta -- but the Arrival field will not necessarily be in order. So that means searching through the column to produce the solution set.

If you use the Arrival field as the Row Key, it will be easy to pull out all flights between March 21st and 29th, but the airport of origin won't be organized. So you will be searching through the arrival column to produce the solution set.

In the third example, a Row Key has been constructed from information extracted from the Origin field and the Arrival field -- creating a constructed Row Key. Because the data is organized lexicographically by the Row Key, all the Atlanta flights will appear in a group, and sorted by date of arrival. Using this Row Key you can generate the solution set with only a scan.

In this example, the data was transformed when it arrived. So constructing a Row Key during the transformation process is straightforward.



Cloud Bigtable schema organization

Column Families

Row Key	Flight_Information					Aircraft_Information			
	Origin	Destination	Departure	Arrival	Passengers	Capacity	Make	Model	Age
ATL#arrival#20190321-1121	ATL	LON	20190321-0311	20190321-1121	158	162	B	737	18
ATL#arrival#20190321-1201	ATL	MEX	20190321-0821	20190321-1201	187	189	B	737	8
ATL#arrival#20190321-1716	ATL	YVR	20190321-1014	20190321-1716	201	259	B	757	23

Google Cloud

Cloud Bigtable also provide Column Families. By accessing the Column Family, you can pull some of the data you need without pulling all of the data from the row or having to search for it and assemble it.
This makes access more efficient.

https://cloud.google.com/bigtable/docs/schema-design