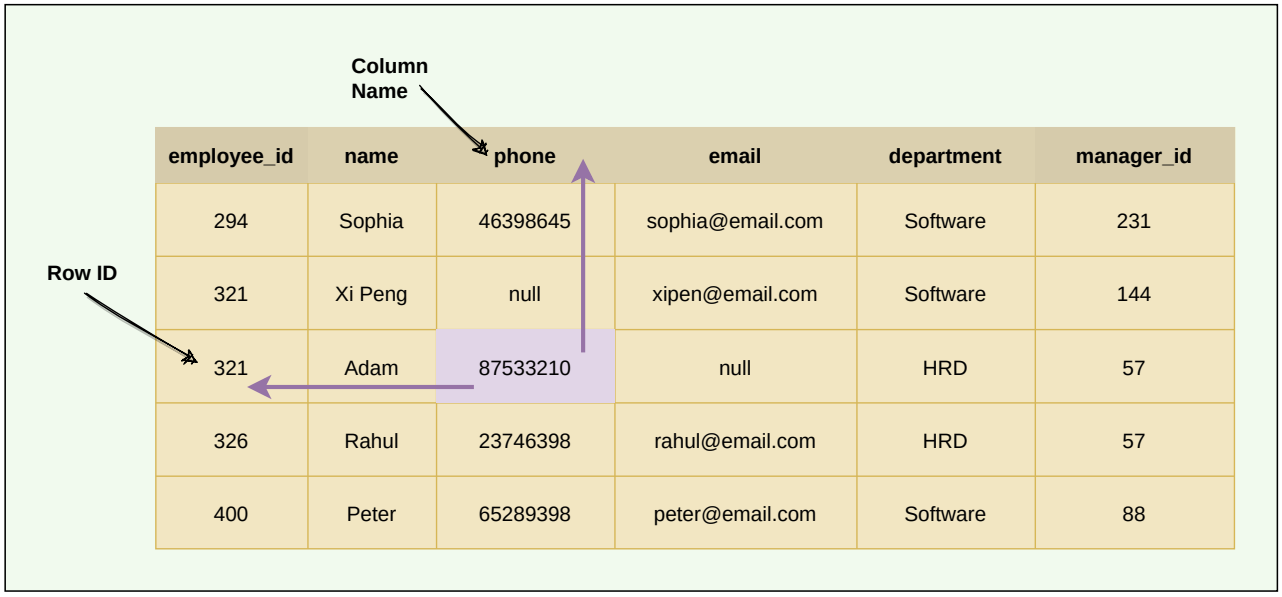
BigTable Data Model

This lesson explains how BigTable models its data.

We'll cover the following

In simple terms, BigTable can be characterized as a sparse, distributed, persistent, multidimensional, sorted map. Let's dig deeper to understand each of these characteristics of BigTable.

Traditional DBs have a two-dimensional layout of the data, where each cell value is identified by the '**Row ID**' and '**Column Name**':

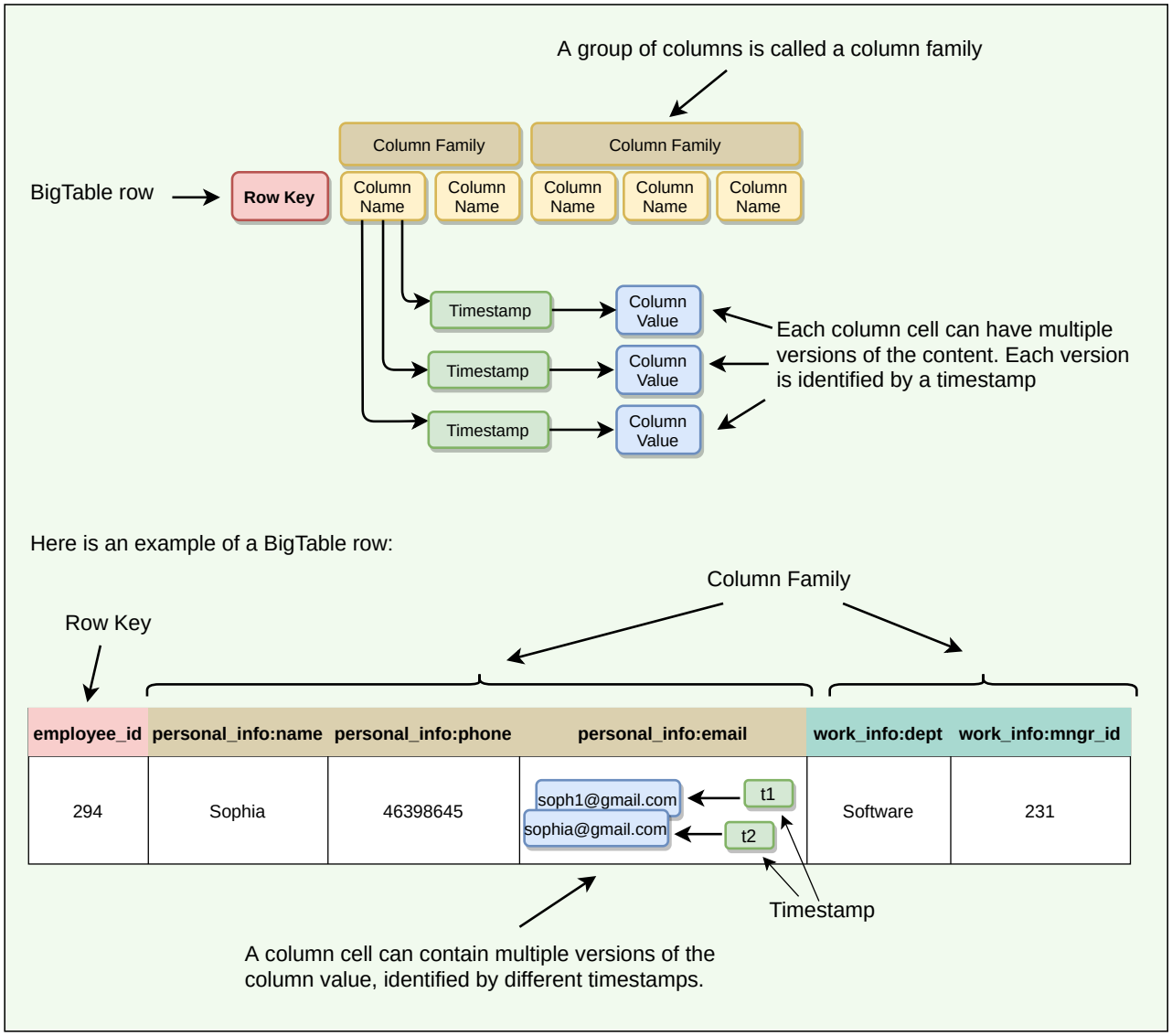


employee_id	name	phone	email	department	manager_id
294	Sophia	46398645	sophia@email.com	Software	231
321	Xi Peng	null	xipen@email.com	Software	144
321	Adam	87533210	null	HRD	57
326	Rahul	23746398	rahul@email.com	HRD	57
400	Peter	65289398	peter@email.com	Software	88

Two-dimensional layout of a traditional database

BigTable has a **four-dimensional data model**. The four dimensions are:

1. Row Key: Uniquely identifies a row
2. Column Family: Represents a group of columns
3. Column Name: Uniquely identifies a column
4. Timestamp: Each column cell can have different versions of a value, each identified by a timestamp



BigTable's four-dimensional data model

The data is indexed (or sorted) by row key, column key, and a timestamp. Therefore, to access a cell's contents, we need values for all of them. If no timestamp is specified, BigTable retrieves the most recent version.

> **(** `row_key` : string, `column_name` : string, `timestamp` : int64 **)** → **cell contents** (string)

Rows#

Each row in the table has an associated row key that is an arbitrary string of up to 64 kilobytes in size (although most keys are significantly smaller):

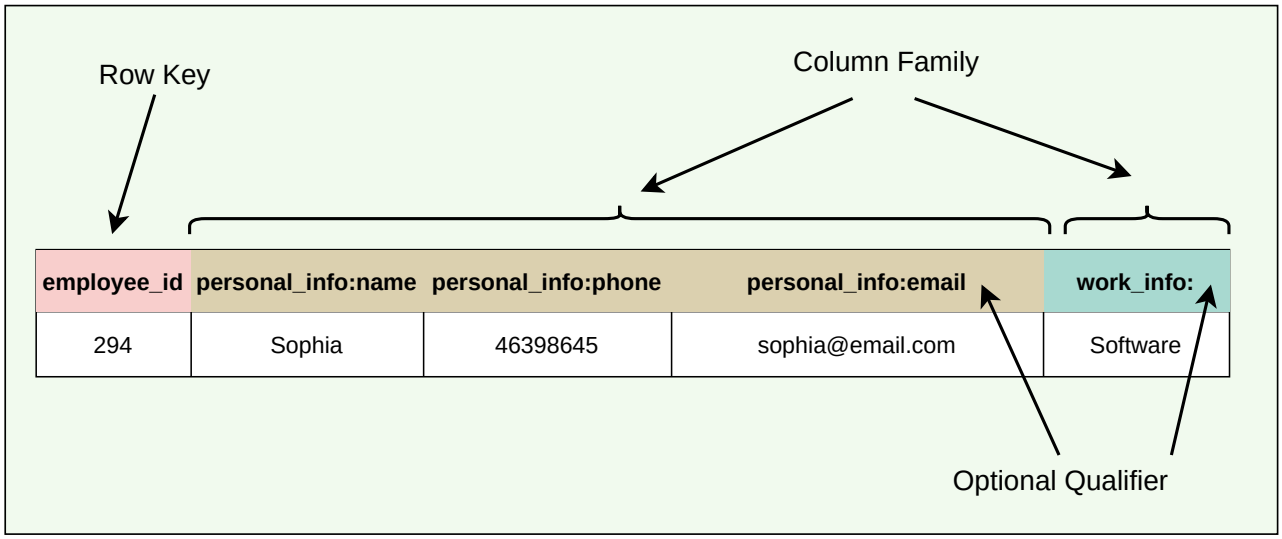
- Each row is uniquely identified by the 'row key.'
- Each 'row key' is internally represented as a string.
- Every read or write of data under a single row is atomic. This also means that atomicity across rows is not guaranteed, e.g., when updating two rows, one might succeed, and the other might fail.
- Each table's data is only indexed by row key, column key, and timestamp. There are no secondary indices.

A **column** is a key-value pair where the key is represented as 'column key' and the value as 'column value.'

Column families#

Column keys are grouped into sets called column families. All data stored in a column family is usually of the same type. The number of distinct column families in a table should be small (in the hundreds at maximum), and families should rarely change during operation. Access control as well as both disk and memory accounting are performed at the column-family level.

The following figure shows a single row from a table. The row key is `294`, and there are two column families: `personal_info` and `work_info`, with three columns under the `personal_info` column family.



Column families

- Column family format: `family:optional qualifier`
- All rows have the same set of column families.
- BigTable can retrieve data from the same column family efficiently.
- Short Column family names are better as names are included in the data transfer.

Columns#

- Columns are units within a column family.
- A BigTable may have an unbounded number of columns.
- New columns can be added on the fly.
- Short column names are better as names are passed in each data transfer, e.g., `ColumnFamily:ColumnName => Work:Dept`
- As mentioned above, BigTable is quite suitable for **sparse data**. This is because empty columns are not stored.

Timestamps#

Each column cell can contain multiple versions of the content. For example, as we saw in the earlier example, we may have several timestamped versions of an employee's email. A 64-bit timestamp identifies each version that either represents real time or a custom value assigned by the client. While reading, if no timestamp is specified, BigTable returns the most recent version. If the client specifies a timestamp, the latest version that is earlier than the specified timestamp is returned.

BigTable supports two per-column-family settings to garbage-collect cell versions automatically. The client can specify that only the last 'n' versions of a cell be kept, or that only new-enough versions be kept (e.g., only keep values that were written in the previous seven days).

← Back

BigTable: Introduction

Next →

System APIs

Mark as Completed

Report an Issue