Explore file storage

5 minutes

The ability to store data in files is a core element of any computing system. Files can be stored in local file systems on the hard disk of your personal computer, and on removable media such as USB drives; but in most organizations, important data files are stored centrally in some kind of shared file storage system. Increasingly, that central storage location is hosted in the cloud, enabling cost-effective, secure, and reliable storage for large volumes of data.

The specific file format used to store data depends on a number of factors, including:

- The type of data being stored (structured, semi-structured, or unstructured).
- The applications and services that will need to read, write, and process the data.
- The need for the data files to be readable by humans, or optimized for efficient storage and processing.

Some common file formats are discussed below.

Delimited text files

Data is often stored in plain text format with specific field delimiters and row terminators. The most common format for delimited data is comma-separated values (CSV) in which fields are separated by commas, and rows are terminated by a carriage return / new line. Optionally, the first line may include the field names. Other common formats include tab-separated values (TSV) and space-delimited (in which tabs or spaces are used to separate fields), and fixed-width data in which each field is allocated a fixed number of characters. Delimited text is a good choice for structured data that needs to be accessed by a wide range of applications and services in a human-readable format.

The following example shows customer data in comma-delimited format:

```
FirstName,LastName,Email
Joe,Jones,joe@litware.com
Samir,Nadoy,samir@northwind.com
```

JavaScript Object Notation (JSON)

JSON is a ubiquitous format in which a hierarchical document schema is used to define data entities (objects) that have multiple attributes. Each attribute might be an object (or a collection of objects); making JSON a flexible format that's good for both structured and semi-structured data.

The following example shows a JSON document containing a collection of customers. Each customer has three attributes (*firstName*, *lastName*, and *contact*), and the *contact* attribute contains a collection of objects that represent one or more contact methods (email or phone). Note that objects are enclosed in braces ({..}) and collections are enclosed in square brackets ([..]). Attributes are represented by *name* : *value* pairs and separated by commas (,).

JSON

```
{
  "customers":
  [
    {
      "firstName": "Joe",
      "lastName": "Jones",
      "contact":
      [
        {
          "type": "home",
          "number": "555 123-1234"
        },
        {
          "type": "email",
          "address": "joe@litware.com"
        }
      ]
    },
    {
      "firstName": "Samir",
      "lastName": "Nadoy",
      "contact":
      [
        {
          "type": "email",
          "address": "samir@northwind.com"
        }
      ]
    }
  ]
}
```

Extensible Markup Language (XML)

XML is a human-readable data format that was popular in the 1990s and 2000s. It's largely been superseded by the less verbose JSON format, but there are still some systems that use XML to represent data. XML uses *tags* enclosed in angle-brackets (<../>) to define *elements* and *attributes*, as shown in this example:

XML

```
<Customers>
  <Customer name="Joe" lastName="Jones">
    <ContactDetails>
      <Contact type="home" number="555 123-1234"/>
      <Contact type="email" address="joe@litware.com"/>
    </ContactDetails>
  </Customer>
  <Customer name="Samir" lastName="Nadoy">
    <ContactDetails>
      <Contact type="email" address="samir@northwind.com"/>
    </ContactDetails>
  </Customer>
</Customers>
```

Binary Large Object (BLOB)

Ultimately, all files are stored as binary data (1's and 0's), but in the human-readable formats discussed above, the bytes of binary data are mapped to printable characters (typically through a character encoding scheme such as ASCII or Unicode). Some file formats however, particularly for unstructured data, store the data as raw binary that must be interpreted by applications and rendered. Common types of data stored as binary include images, video, audio, and application-specific documents.

When working with data like this, data professionals often refer to the data files as *BLOBs* (Binary Large Objects).

Optimized file formats

While human-readable formats for structured and semi-structured data can be useful, they're typically not optimized for storage space or processing. Over time, some specialized file formats that enable compression, indexing, and efficient storage and processing have been developed.

Some common optimized file formats you might see include *Avro*, *ORC*, and *Parquet*:

- *Avro* is a row-based format. It was created by Apache. Each record contains a header that describes the structure of the data in the record. This header is stored as JSON. The data is stored as binary information. An application uses the information in the header to parse the binary data and extract the fields it contains. Avro is a good format for compressing data and minimizing storage and network bandwidth requirements.
- *ORC* (Optimized Row Columnar format) organizes data into columns rather than rows. It was developed by HortonWorks for optimizing read and write operations in Apache Hive (Hive is a data warehouse system that supports fast data summarization and querying over large datasets). An ORC file contains *stripes* of data. Each stripe holds the data for a column or set of columns. A stripe contains an index into the rows in the stripe, the data for each row, and a footer that holds statistical information (count, sum, max, min, and so on) for each column.
- *Parquet* is another columnar data format. It was created by Cloudera and Twitter. A Parquet file contains row groups. Data for each column is stored together in the same row group. Each row group contains one or more chunks of data. A Parquet file includes metadata that describes the set of rows found in each chunk. An application can use this metadata to quickly locate the correct chunk for a given set of rows, and retrieve the data in the specified columns for these rows. Parquet specializes in storing and processing nested data types efficiently. It supports very efficient compression and encoding schemes.

Next unit: Explore databases

Continue >