**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:

Copy

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 (**,**).

JSONCopy

{

"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:

XMLCopy

<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

**Khám phá Lưu trữ File**

Khả năng lưu trữ dữ liệu trong các file là một yếu tố cốt lõi của bất kỳ hệ thống máy tính nào. Các file có thể được lưu trữ trong hệ thống file cục bộ trên ổ cứng của máy tính cá nhân của bạn và trên các thiết bị lưu trữ di động như ổ đĩa USB; nhưng ở hầu hết các tổ chức, các file dữ liệu quan trọng được lưu trữ tập trung trong một hệ thống lưu trữ file共享 (shared) nào đó. Ngày càng nhiều, vị trí lưu trữ trung tâm đó được lưu trữ trên đám mây, cho phép lưu trữ hiệu quả về chi phí, an toàn và đáng tin cậy cho khối lượng dữ liệu lớn.

Định dạng file cụ thể được sử dụng để lưu trữ dữ liệu phụ thuộc vào một số yếu tố, bao gồm:

* Loại dữ liệu được lưu trữ (có cấu trúc, bán cấu trúc hoặc không cấu trúc).
* Các ứng dụng và dịch vụ cần đọc, ghi và xử lý dữ liệu.
* Nhu cầu các file dữ liệu có thể đọc được bởi người dùng hoặc được tối ưu hóa để lưu trữ và xử lý hiệu quả.

Một số định dạng file phổ biến được thảo luận dưới đây.

**Các file văn bản phân cách (Delimited text files)**

Dữ liệu thường được lưu trữ ở định dạng văn bản thuần túy với các dấu phân cách trường (field delimiter) và ký hiệu kết thúc dòng (row terminator) cụ thể. Định dạng phổ biến nhất cho dữ liệu phân cách là giá trị được phân cách bằng dấu phẩy (CSV - comma-separated values) trong đó các trường được phân cách bằng dấu phẩy và các dòng được kết thúc bằng dấu xuống dòng (carriage return). Ngoài ra, dòng đầu tiên có thể bao gồm tên các trường. Các định dạng phổ biến khác bao gồm giá trị được phân cách bằng tab (TSV - tab-separated values) và được phân cách bằng dấu cách (space-delimited) (trong đó các tab hoặc dấu cách được sử dụng để phân cách các trường) và dữ liệu có chiều rộng cố định (fixed-width data) trong đó mỗi trường được分配 (allocation - phân bổ) một số lượng ký tự cố định. Văn bản phân cách là một lựa chọn tốt cho dữ liệu có cấu trúc cần được truy cập bởi nhiều ứng dụng và dịch vụ khác nhau ở định dạng có thể đọc được bởi người dùng.

Ví dụ sau đây hiển thị dữ liệu khách hàng ở định dạng phân cách dấu phẩy:

Họ,Tên,Email

Nam,Nguyễn,nam.nguyen@congty.com

Hoa,Trần,hoa.tran@congtykhac.com

**JavaScript Object Notation (JSON)**

JSON là một định dạng phổ biến trong đó lược đồ tài liệu phân cấp được sử dụng để định nghĩa các thực thể dữ liệu (đối tượng) có nhiều thuộc tính. Mỗi thuộc tính có thể là một đối tượng (hoặc một tập hợp các đối tượng); biến JSON thành một định dạng linh hoạt phù hợp cho cả dữ liệu có cấu trúc và bán cấu trúc.

Ví dụ sau đây hiển thị một tài liệu JSON chứa một tập hợp khách hàng. Mỗi khách hàng có ba thuộc tính (firstName, lastName và contact), và thuộc tính contact chứa một tập hợp các đối tượng đại diện cho một hoặc nhiều phương thức liên lạc (email hoặc điện thoại). Lưu ý rằng các đối tượng được bao trong ngoặc nhọn ({..}) và các tập hợp được bao trong ngoặc vuông ([..]). Thuộc tính được biểu diễn bởi các cặp name:value và được phân cách bằng dấu phẩy (,).

JSON

{

"customers":

[

{

"firstName": "Nam",

"lastName": "Nguyen",

"contact":

[

{

"type": "nhà riêng",

"number": "024 3856 1234"

},

{

"type": "email",

"address": "nam.nguyen@congty.com"

}

]

},

{

"firstName": "Hoa",

"lastName": "Tran",

"contact":

[

{

"type": "email",

"address": "hoa.tran@congtykhac.com"

}

]

}

]

}

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

**Ngôn ngữ đánh dấu mở rộng (Extensible Markup Language - XML)**

XML là một định dạng dữ liệu có thể đọc được bởi người dùng, phổ biến vào những năm 1990 và 2000. Nó phần lớn đã được thay thế bởi định dạng JSON ít冗长 (r冗長 -冗贅 - dài dòng,囉嗦 - la oai

**Định dạng XML:**

XML

<Customers>

<Customer name="Nam" lastName="Nguyen">

<ContactDetails>

<Contact type="nhà riêng" number="024 3856 1234"/>

<Contact type="email" address="nam.nguyen@congty.com"/>

</ContactDetails>

</Customer>

<Customer name="Hoa" lastName="Tran">

<ContactDetails>

<Contact type="email" address="hoa.tran@congtykhac.com"/>

</ContactDetails>

</Customer>

</Customers>

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

**Đối tượng Lớn Nhị phân (Binary Large Object - BLOB)**

Cuối cùng, tất cả các file đều được lưu trữ dưới dạng dữ liệu nhị phân (1 và 0), nhưng trong các định dạng có thể đọc được bởi người dùng được thảo luận ở trên, các byte dữ liệu nhị phân được ánh xạ đến các ký tự có thể in (thường thông qua một lược đồ mã hóa ký tự như ASCII hoặc Unicode). Tuy nhiên, một số định dạng file, đặc biệt là đối với dữ liệu không cấu trúc, lưu trữ dữ liệu dưới dạng nhị phân thô mà các ứng dụng phải giải thích và hiển thị. Các loại dữ liệu phổ biến được lưu trữ dưới dạng nhị phân bao gồm hình ảnh, video, âm thanh và tài liệu ứng dụng.

Khi làm việc với dữ liệu như thế này, các chuyên gia dữ liệu thường gọi các file dữ liệu là BLOB (Binary Large Objects).

**Định dạng file tối ưu hóa:**

Mặc dù các định dạng có thể đọc được bởi người dùng cho dữ liệu có cấu trúc và bán cấu trúc có thể hữu ích, nhưng chúng thường không được tối ưu hóa cho không gian lưu trữ hoặc xử lý. Theo thời gian, một số định dạng file chuyên biệt cho phép nén, lập chỉ mục và lưu trữ và xử lý hiệu quả đã được phát triển.

**Một số định dạng file tối ưu hóa phổ biến bạn có thể thấy bao gồm Avro, ORC và Parquet:**

* **Avro:** Là định dạng dựa trên hàng. Nó được tạo ra bởi Apache. Mỗi bản ghi chứa một tiêu đề mô tả cấu trúc của dữ liệu trong bản ghi. Tiêu đề này được lưu trữ dưới dạng JSON. Dữ liệu được lưu trữ dưới dạng thông tin nhị phân. Một ứng dụng sử dụng thông tin trong tiêu đề để phân tích cú pháp dữ liệu nhị phân và trích xuất các trường có trong đó. Avro là một định dạng tốt để nén dữ liệu và giảm thiểu yêu cầu về băng thông lưu trữ và mạng.
* **ORC (Optimized Row Columnar format):** Sắp xếp dữ liệu thành các cột thay vì các hàng. Nó được phát triển bởi HortonWorks để tối ưu hóa các hoạt động đọc và ghi trong Apache Hive (Hive là một hệ thống kho dữ liệu hỗ trợ tóm tắt dữ liệu nhanh và truy vấn trên các tập dữ liệu lớn). Một file ORC chứa các dải dữ liệu. Mỗi dải chứa dữ liệu cho một cột hoặc tập hợp các cột. Một dải chứa một chỉ mục vào các hàng trong dải, dữ liệu cho mỗi hàng và một phần chân trang chứa thông tin thống kê (số lượng, tổng, tối đa, tối thiểu, v.v.) cho mỗi cột.
* **Parquet:** Là một định dạng dữ liệu cột khác. Nó được tạo ra bởi Cloudera và Twitter. Một file Parquet chứa các nhóm hàng. Dữ liệu cho mỗi cột được lưu trữ cùng nhau trong cùng một nhóm hàng. Mỗi nhóm hàng chứa một hoặc nhiều phần dữ liệu. Một file Parquet bao gồm siêu dữ liệu mô tả tập hợp các hàng được tìm thấy trong mỗi phần. Một ứng dụng có thể sử dụng siêu dữ liệu này để nhanh chóng xác định vị trí phần chính xác cho một tập hợp các hàng nhất định và truy xuất dữ liệu trong các cột được chỉ định cho các hàng này. Parquet chuyên lưu trữ và xử lý các kiểu dữ liệu lồng nhau một cách hiệu quả. Nó hỗ trợ các lược đồ nén và mã hóa rất hiệu quả.