2.1.5 Data Object (DO)

Return to Common Definitions

A **Data Object (DO)** is a collection of one or more pieces of data that when aggregated together impart meaning to the whole. In other words, **Data Object** is an alternate way of saying “this group of data should be considered as one piece of data.”

A common example of DOs are data tables where each row represents one Data Object and the columns represent the pieces of data that comprise the object. Some other examples of DOs include arrays, structures, collection of other Data Objects, and scalar types (i.e., not the value but rather the type).

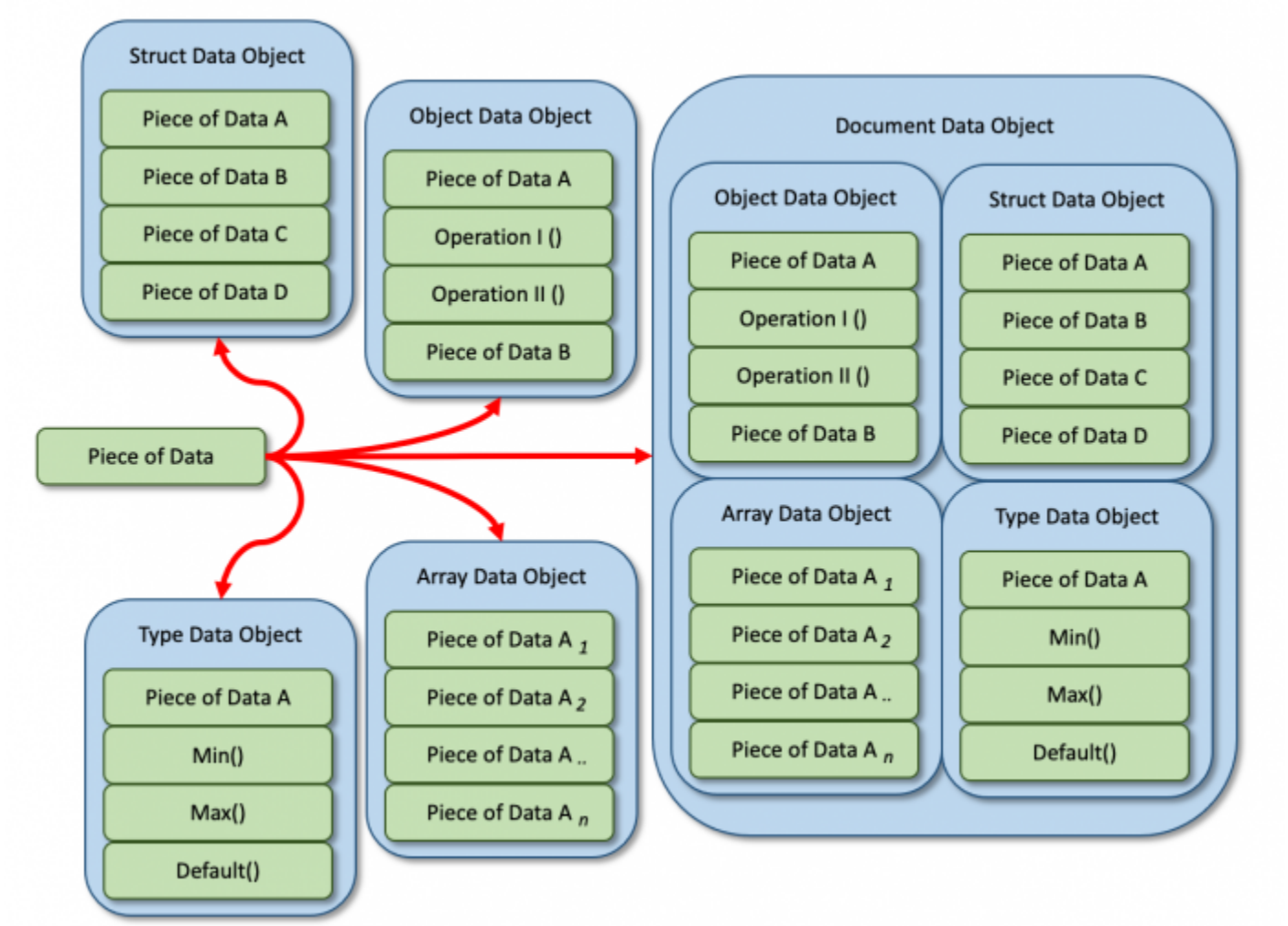

Figure 1: Examples of Data Objects.

Another way to conceptualize a “data object” is as a way of compartmentalizing information into smaller, reusable pieces that are easier to understand and analyze with Subject Matter Experts (SMEs) usually being responsible for the definitions. It is a good idea to model Data Objects as concepts in Unified Modeling Language (UML) or System Modeling Language (SysML).

Different Datastore Architectures

Return to Top

Data Objects are stored in memory or permanent storage using any number of architectures. In addition, regardless of how a Data Object is stored in Memory or on permanent storage, while in transit the data may use a different architecture. For example, a Relational DataBase Management System (RDBMS) may be transmitted over the wired using a set of key-value pairs such as JSON.

There is no single internal organization of a Data Object and it depends on the implementor of the Data Object to select and use the models that work best for the intended purpose of the Data Object. Some common models described by Grant[1)] are:

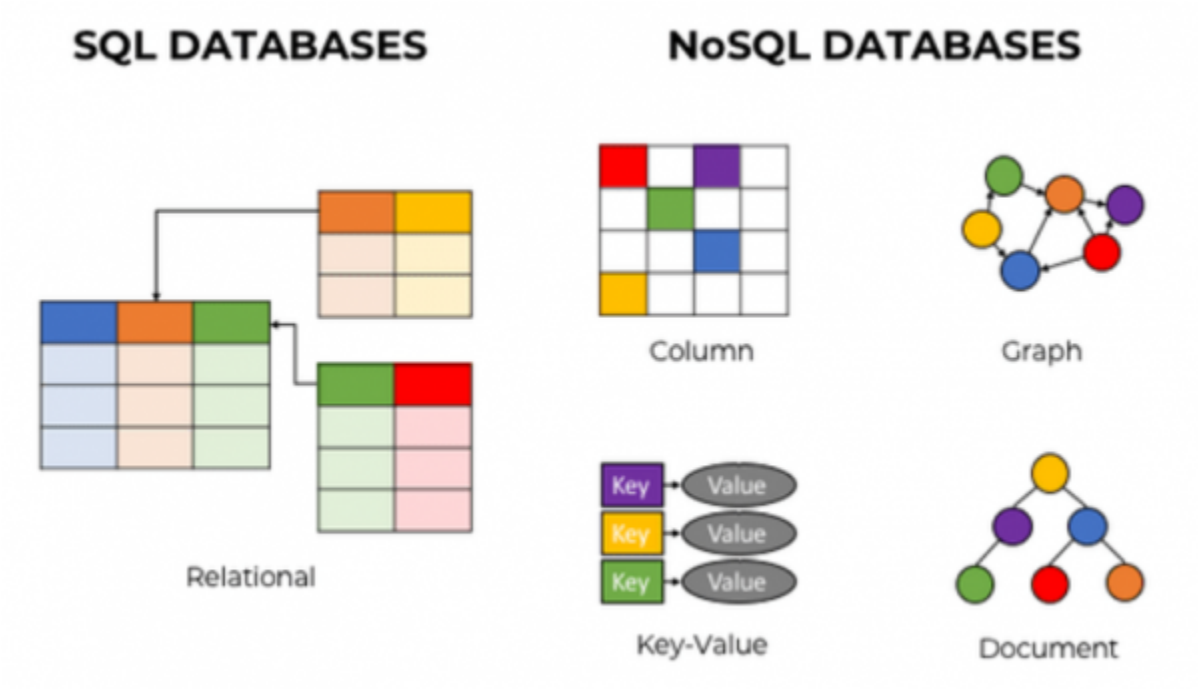

Figure 2: The Types of Datastore (See Grant).

SQL Datastores

Return to Top

SQL Datastores are built around Structured Query Language (SQL). Most SQL Datastores are Relational DataBase Management System (RDBMS), which store data that is primarily structured. SQL has been in existence since the 1970s and is standardized for use across many RDBMS systems. (See Grant)
Relational datastores

> Relational datastores are tabular in that they store data in tables with rows and columns. The tables are interconnected through columns shared in common between tables. In the above image, the arrows show the columns that connect the data in various tables. SQL Datastores are particularly well suited for systems dealing with transactions, such as customer relations

management tools and accounting software. They are not, however, structured in such a manner as to "play well" with [[dido:public:ra:xapend:xapend.a_glossary:o:oop|object-oriented programming]] languages, necessitating object-relational mapping tools. (See Grant)

NoSQL Datastores

Return to Top

NoSQL Datastores, as the name implies, generally do not rely on SQL, though some may use SQL-like languages. Some, in fact, say NoSQL stands for “Not only SQL.” NoSQL Datastores are generally less structured, more diverse, and newer overall than their SQL counterparts. Some common examples are listed below. (See Grant)

Wide Column	Wide Column Stores are similar to SQL Datastores in that they use tables of rows and columns (i.e., Data Model). The columns, however, can change with each row, creating a 2-dimensional effect.(See Grant)
Key-Value	Key-Value Datastores are essentially hash tables, both of which can be compared to a dictionary, in which one key (a word entry) corresponds to many underlying fields of information. The underlying fields for the keys in Key-Value Datastores can vary from key to key, lending the system a great deal of flexibility in the data it can store. The key-value-pair system matches well with object-oriented programming, and since a key's values are not predetermined with placeholders as in a table, Datastores of this type perform very well in certain circumstances.(See Grant)
Graph Data Model	Graph Datastores are suited for data that is highly interconnected, because of the “relationships” (also known as “edges”) it maintains between nodes. As a result, complex data lookup can perform very highly because of the existing connections between data. Since SQL Datastores implicitly compare tables of highly ordered data, SQL is poorly suited to the nature of Graphs, in which the relationships are a concrete part of the data structure. Graphs, in fact, do not have a standardized language, and usually rely on a proprietary system.(See Grant)
Document Data Model	Document-Oriented Datastores are one of the most prominent types of non-relational Datastores and are actually a variety of Key-Value Datastores. They differ mainly in the way the Datastores engine processes the data, rather than their structure. Documents are very closely related to objects in programming languages, and documents are very flexible. They can have any number of fields and fields can vary between documents. Documents themselves each have a unique key used to retrieve the documents themselves, and the keys often have an index to hasten lookup. (See Grant)

DIDO Data Object Datastores

Return to Top

The mechanism used to store a Data Object in a properly architected, designed, and implement DIDO should not be of concern to the user of the DIDO. Those details should be abstracted from the DIDO user encapsulation and the use of APIs. However, how the data is transmitted from within the DIDO to the outside is of importance. The current DIDO industry trend is to use **Key-Value** pairs where the **Value** is another Data Object or a primitive Data Type (i.e., atom). This allows the data to be transmitted

relatively efficiently (still as text); be human-readable; flexible and dynamic in nature. Many modern programming languages require little to no processing by the programmers to access the Key-Value pairs in the data. For example, Javascript inherently constructs JavaScript Object Notation (JSON) Objects for transmission or more permanent storage and allows the objects to be easily realized when they are received in transmission or read from storage.

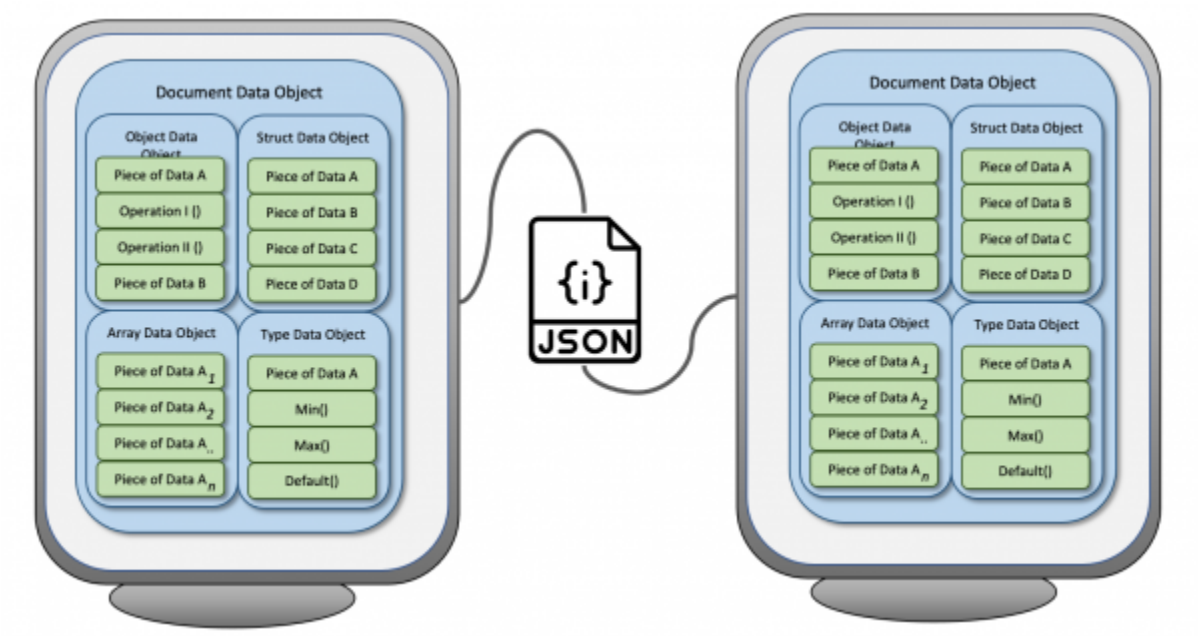

Figure 3: Data in Motion often uses JSON

Originally, eXtensible Markup Language (XML) was used to accomplish the same thing as JSON, but XML required more data to represent the same information and the manipulation of the XML Documents was not as “natural” or as “native” to the programmers as the recreated objects from JSON.

Many programming languages (i.e., PHP, Java, C#, C++, Rust, Solidity, etc) and frameworks have adopted JSON as a standard way to transmit and receive data. This makes Interoperability and integration easier, since both ends of the “pipe” can be written in languages that are best suited for the task at hand.

DIDO Data Objects in Relational Database

Return to Top

Keep in mind, Ethereum is the database, smart contracts are the data tables, and transactions from wallets are the rows in each table.[2)]

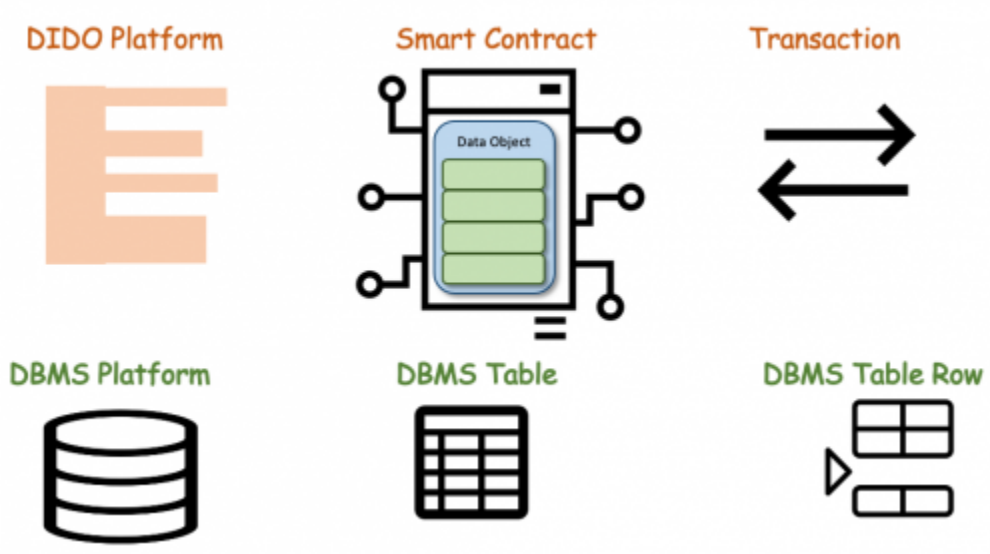

Figure 4:

1)
Grant, dev.to, 27 July 2020, Database Types Explained, Accessed on: 22 May 2021, https://dev.to/gduple/database-types-explained-4ja7
2)
Andrew Hong, Hands-On Tutorials, Your guide to basic SQL while learning Ethereum at the same time, 17 April 2021, Accessed: 11 June 2021, https://towardsdatascience.com/your-guide-to-basic-sql-while-learning-ethereum-at-the-same-time-9eac17a05929