Notes on Physical & Logical Data Layouts

Michael Hausenblas[1]

MapR Technologies EMEA, Galway, Ireland
mhausenblas@maprtech.com

Abstract. In this short note I review and discuss fundamental options for physical and logical data layouts as well as the impact of the choices on data processing. I should say in advance that these notes offer no new insights, that is, everything stated here has already been published elsewhere. In fact, it has been published in so many different places, such as blog posts, in the literature, etc. that the main contribution is to bring it all together in one place.

1 Motivation

Data processing and management systems such as databases, datastores [Cat11] or query engines usually have to answer to two kinds of entities: *humans* and *hardware.*

Towards humans, they provide means to query, manipulate or manage[1] data. Towards the hardware, they issue store and retrieve commands. They depend directly or indirectly on the very nature of the hardware. Almost all systems— for example, Hadoop's distributed file system—are designed with strong though not necessarily explicit assumptions about the underlying hardware such as hard disk drives (HDD) [Ele09], their spindles, heads, etc.

Conceptually, there are three levels present in data processing and management systems (Fig. 1):

– The *User Interface* level. Any database or datastore needs to provide a way to interact with the data under management. This can be something elaborate, standardised and mature as the Structured Query Language (SQL) found in relational database management systems (RDBMS), such as Oracle DB, PostgreSQL, or MySQL. This can be a RESTful interface, found in many NoSQL datastores, like, for example, CouchDB's API[2]. Of course, this can also be a programming-language-level API such as the case with Hadoop[3].

---

[1] The management aspect can span a wide range of activities including but not limited to snapshots, mirroring, etc.

[2] See documentation at http://wiki.apache.org/couchdb/HTTP_Document_API

[3] http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/package-summary.html
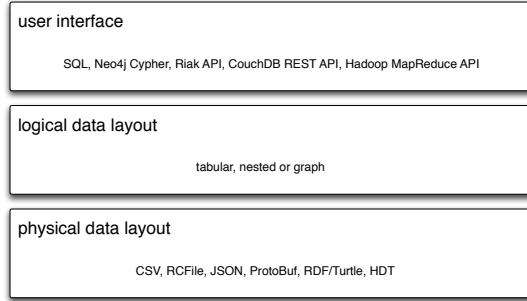
Fig. 1. The three levels of data representation and interaction in data management systems, including examples for each of the levels.

- The *Logical Data Layout* level. Addresses how the user conceptually thinks about and deals with the data. In case of a RDBMS the data units might be tables and records, in a key-value store like Redis it may be an entry identified via a key and in a wide-column store the data unit might be a row containing different columns, and last but not least in an RDF store a single triple might be the unit one logically manipulates.
- The *Physical Data Layout* level. On this level, we're concerned with the question how the data is laid out once serialised. The serialisation takes place from main memory (RAM) either to send the data in question over the wire, or, to be stored on a durable medium such as a hard disk drive or a solid-state drive (SSD) [Cor12]. Concrete serialisations may be textual based, such as CSV and JSON or of binary nature, like the RCFile format [HLH$^+$11].

2 Manifestations of Data Layouts

In [HVTC12] we introduced the three fundamental data shapes *tabular*, *tree*— henceforth *nested*, and *graph*. It turns out that it is useful to further differentiate the shapes, distinguishing between logical and physical layouts, as hinted above.

In the following, I propose a non-exhaustive, lightweight taxonomy for logical and physical data layouts and serialisation formats as depicted in Fig. 2.

The main point of this taxonomy is to decouple the logical from the physical level. While for the human user the logical level is of importance, from a software and systems perspective the physical level dominates. There are cases, however, where the abstraction is leaking and the user is forced to accommodate.

Take, for example, best practices concerning NoSQL data modeling[4]: with a wide-column store, such as HBase, one can easily get into a situation where

---

[4] As found in the blog post "NoSQL Data Modeling Techniques" via `http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/`
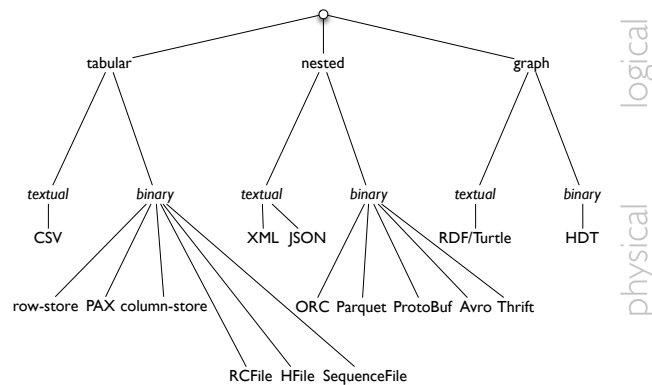
Fig. 2. A non-exhaustive, lightweight taxonomy for logical and physical data layouts and serialisation formats commonly used in the data processing community.

one must take into account the physical location of the data in order to avoid performance penalties[5].

Also, the choice of the serialisation format (for example, textual vs. binary) can have severe implications, both in terms of performance and maintenance. Look at a case where one decides to use JSON as the wire format in contrast to, say, Avro. In the former case, one can debug any document simply by issuing a command on the shell like `cat datafile.json | more` while with Avro more specialised tooling is necessary. On the other hand, one can probably expect a better I/O performance and disk utilisation with a binary format such as Avro, compared to JSON.

Now we're already entering the discussion of the impact of choices we make concerning how the data is laid out. Let's jump right into it.

3   Impact on Data Processing at Scale

There are two schools of thought concerning the organisation of data units: data *normalisation*, and data *denormalisation*. The former wants to minimise redundancy, the latter aims to minimise assembly. Both have their own built-in assumptions, characteristics and use cases:

Normalised data . . .

– As data items are not redundant, data consistency is relatively easy to achieve compared to denormalised data.
– When updating data in place one only has to deal with it once and not in multiple locations.
– Storage is efficiently used, that is, it takes up less disk space.

---

[5] http://stackoverflow.com/questions/10806955/hbase-schema-key-for-real-time-analytics-solution

Denormalised data . . .

- The access to data units is fast as no joins are necessary; the data can be considered to be pre-joined.
- As it provides an entity-centric view, it is in general more straight-forward to employ automated sharding of the data.
- Due to keeping multiple copies of data items or fragments thereof around, it requires typically a multitude more space on disk than normalised data.

In Table 1 I'm providing a comparison and summary of the two different ways to handle data including typical examples of workloads and technologies concerning use cases.

Table 1. A comparison of normalised vs. denormalised handling of data on the logical and physical level across SQL and NoSQL data management systems.

	NORMALISED	DENORMALISED
characteristics	Each data item is stored exactly in one place.	The data items are repeated as needed.
advantages	Built-in data consistency and storage-efficiency.	Fast, entity-centric data access without the need for joins.
disadvantages	Joints are costly and hard to implement (especially distributed).	Inflexible and storage-hungry.
workloads	OLTP, write-intensive	OLAP, read-intensive
examples	Classical, textbook relational database modelling	Wide-column datastores (HBase, Cassandra), document-oriented datastores (MongoDB, CouchDB), key-value datastores (Redis, Memcached, etc.), graph databases (Neo4j, RDF stores), large-scale relational databases

Allow me a side remark relative to the ongoing and tiring debate SQL vs. NoSQL: it turns out that the focus on SQL as the representative of the evil is really a rather backward view. As stated in many places all over the Web[6] many open source projects and commercial entities are introducing SQL bindings or interfaces on top of Hadoop and NoSQL datastores.

This is quite understandable, given the huge number of deployed (business intelligence) tools that natively speak SQL and of course the many people out there trained in this language.

---

[6] For example, see the blog post `http://gigaom.com/2013/02/21/sql-is-whats-next-for-hadoop-heres-whos-doing-it/`

Joining the dots. We are now in a position to wrap up on the impact of choices we make concerning how the data is laid out: one dimension of freedom is the choice how we organise the data: normalised vs. denormalised. The second choice we have is concerning the physical data representation. Interestingly, some systems are more rigid and upfront with what they support, expect or allow. While, for example, in the Hadoop ecosystem it is entirely up to you how you serialise your data—and depending on your requirements and the workload you might end up with a different result—traditional RDBMS are much more restrictive. Seldom you get to choose the physical data layout and the logical layout is hard-coded anyways.

Coming back full circle to the initial Fig. 1 one should, however, not underestimate the *User Interface* level. At the end of the day the usability, integrability and user familiarity of this level can be the reason why some data management systems may have a better chance to survive than others. Last but not least, one should take into account the emerging *Polyglot Persistence*[7] meme that essentially states that one size does not fit it all concerning data storage and manipulation. I suggest embracing this meme together with Pat Helland's advice [Hel11]: "In today's humongous database systems, clarity may be relaxed, but business needs can still be met."

4   Acknowledgements

References

Cat11.   Rick Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Rec.*, 39:12–27, 2011.

Cor12.   Michael Cornwell. Anatomy of a solid-state drive. *Commun. ACM*, 55(12):59–63, 2012.

Ele09.   Jon Elerath. Hard Disk Drives: The Good, the Bad and the Ugly! *Commun. ACM*, 52(6):38–45, 2009.

Hel11.   Pat Helland. If You Have Too Much Data, then "Good Enough" Is Good Enough. *ACM Queue*, 9:40:40–40:50, 2011.

HLH+11.   Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *ICDE*, pages 1199–1208. IEEE Computer Society, 2011.

HVTC12.   Michael Hausenblas, Boris Villazon-Terrazas, and Richard Cyganiak. Data Shapes and Data Transformations. *arXiv*, 1211.1565, 2012.

---

[7] `http://martinfowler.com/bliki/PolyglotPersistence.html`