

Article development led by [acmqueue](http://queue.acm.org)
queue.acm.org

In today's humongous database systems, clarity may be relaxed, but business needs can still be met.

BY PAT HELLAND

If You Have Too Much Data, then 'Good Enough' Is Good Enough

CLASSIC DATABASE SYSTEMS offer crisp answers for a relatively small amount of data. These systems hold their data in one or a relatively small number of computers. With a tightly defined schema and transactional consistency, the results returned from queries are crisp and accurate.

New systems have humongous amounts of data content, change rates, and querying rates and take lots of computers to hold and process. The data quality

and meaning are fuzzy. The schema, if present, is likely to vary across the data. The origin of the data may be suspect, and its staleness may vary.

Today's data systems coalesce data from many sources. The Internet, B2B, and enterprise application integration (EAI) combine data from different places. No computer is an island. This large amount of interconnectivity and interdependency has led to a relaxation of many database principles. Let's consider the ways in which today's answers differ from what we used to expect.

The Erosion of Principles

Many different principles of classic SQL databases are being eroded by combining too much data:

- ▶ *Unlocked data.* Classic SQL semantics depend on the data in the database being locked. Unlocking the data changes the semantics from the classic database.

- ▶ *Inconsistent schema.* Different documents come from different sources. You need to deal with extensibility, different semantics, and unknown semantics.

- ▶ *Extract, transform, and load.* Data may come from many sources, and you are attempting to shoehorn it into something resembling commonality.

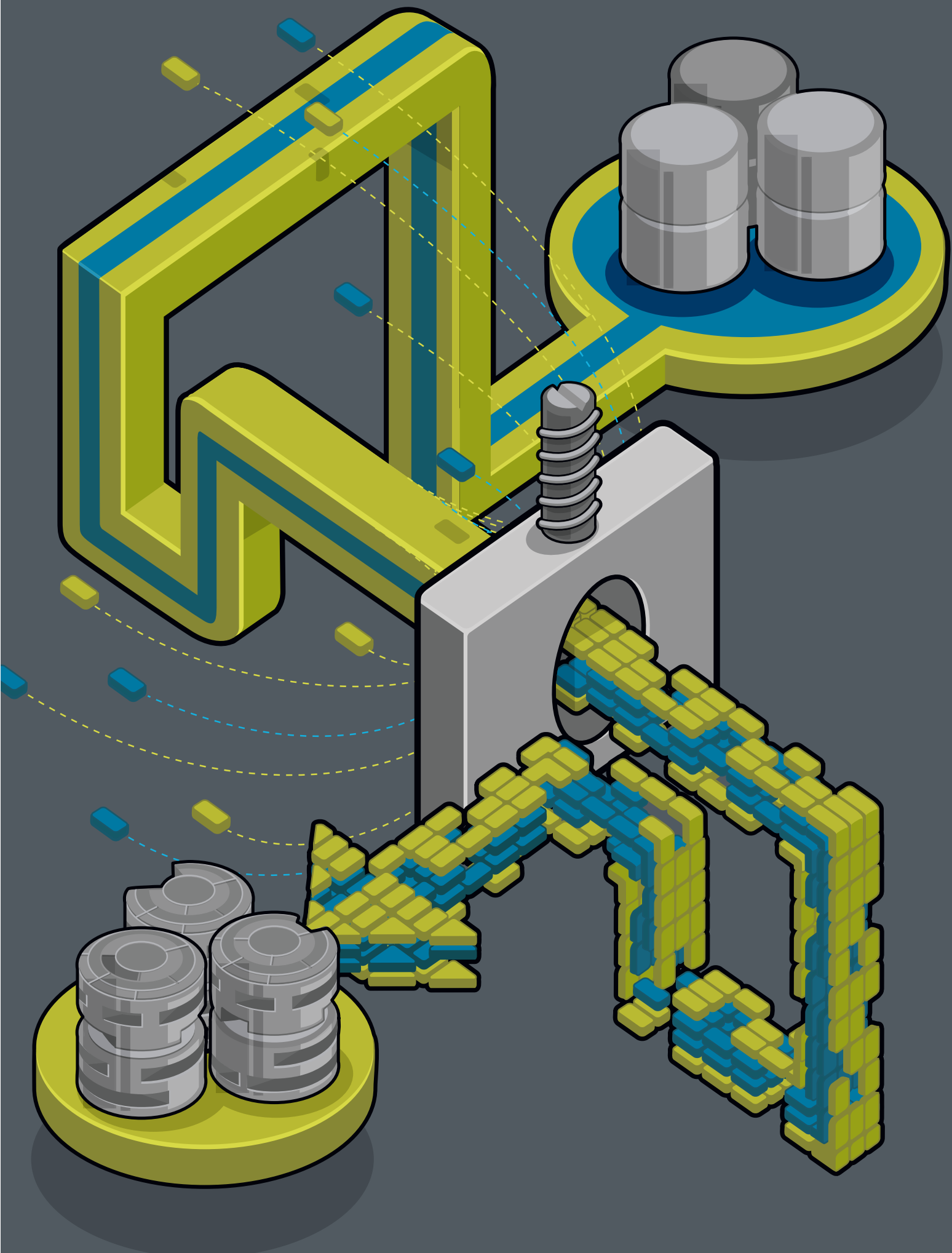
- ▶ *Patterns by inference.* Where are the connections that you didn't think of? What can you infer by continuously examining the data?

- ▶ *Too much to be accurate.* By the time you do the calculations, the answer may have changed. Too much, too fast—you need to approximate.

Business needs lead to lossy answers. While we would like to be perfect, business needs answers even when the answers are not perfect.

- ▶ *Sometimes, the data causes challenges.* There may be a huge volume of data; it may be from many different sources or unclear sources; it may arrive over a period of time.

- ▶ *Sometimes, the processing causes challenges.* You may need to convert, transform, or interpret the data in a way that loses information (see Figure



1). You may need to make inferences and assumptions about the data.

We can no longer pretend to live in a clean world. SQL and its Data Definition Language (DDL) assume a crisp and clear definition of the data, but that is a subset of the business examples we see in the world around us. It's OK if we have lossy answers—that's frequently what business needs.

Einstein Was on to Something

Huge scale has implications on the consistency semantics of the data. It turns out that most large systems in the NoSQL (not only SQL) genre cannot support transactions across all of their data. This carries implications on how data can be used together. Here, I examine some of the issues with data

from distant systems.

Transactions, classic databases, and “now.” Transactions make you feel alone. The purpose of a database transaction is to give the appearance that nothing else is changing the data while you are doing your work.

As a transaction geek, I've spent decades working on systems that provide a guaranteed form of consistency. One special form of this consistency is called *serializability*, or the appearance of a serial order. The individual transactions do not have to execute in serial order, but there must be at least one serial order that corresponds to the observed behavior.

Each transaction shown in Figure 2 seems to be executing in a crisp and clear “now.” Some stuff happened in the

past; other stuff happens in the future; the transaction sees “now.” The definition of “now” is the boundary within which a transaction can be applied.

SQL databases depend on living in the “now” to provide their semantics. Though many SQL systems offer relaxed consistency semantics to increase concurrency, transaction serializability is still the clearest way to think about the general SQL mechanisms.

Unlocked data is not “now.” Messages and documents contain unlocked data that has been sent out into the wild, cruel world. When messages are sent, they usually cross boundaries that can support distributed transactions; hence, the data is not transaction protected. Instead, the data is typically given an identity and a version. The data from the originating system may change shortly after the message is sent. Unlocking it allows change. Messages and documents are always from the past.

Simultaneity does not exist at a distance. Knowledge travels at the speed of light. By the time you see a distant object in the night sky, it may have changed. Similarly, by the time you receive a message from a distant computer, the data contained in that system may have changed.

Transactions and application/business boundaries define simultaneity. Inside a single system (either one computer or a small number of computers) you can have a transactional database with transaction-protected application operations. Many single applications are growing so large that the implementation of these as a single transactional domain is impractical. Hence, we are now seeing applications using NoSQL-style data in which different portions of an application's data live within different transactional boundaries. Some recent scalable systems offer a small collection of data within which you have transactional consistency. Across these boundaries, looser consistency is provided and the application is responsible for application-specific consistency. These include Google's Megastore¹ and Microsoft's SQL Azure.⁴

Unlocked data: A blast from the past. All data on the Internet is from the “past.” By the time you see it, the truthful state of any changing values may be different. Each independent database,

Figure 1. Data transformation is frequently a lossy operation.

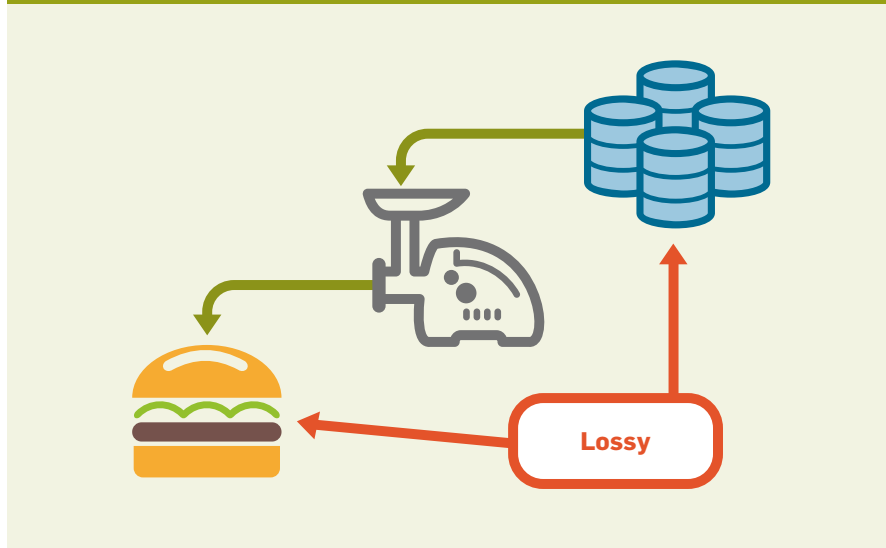
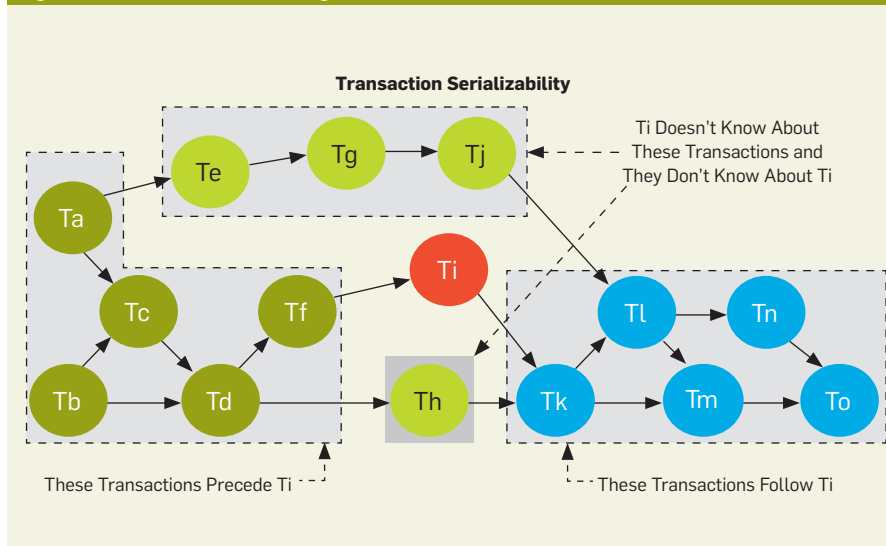


Figure 2. Transactions executing in the “now.”



system, or application will cultivate its own data, which will change over time. In loosely coupled systems, each system has a “now” inside and a “past” arriving in messages. By the time you’ve seen some unlocked message or document, the truth may have changed.

If you depend on a set of data being cohesive, it must be written by its source as a single identity within a single atomic transaction. This cohesive data unit must have a unique identity and version. In today’s large, loosely coupled systems, there will be many of these sets of data. Within each, the temporal nature can be cohesive. Across them, the application must reason about their different histories.

Application semantics must deal with “now” and “then.” Each application running in this large and loosely connected world must have its own subjective sense of “now,” and it must understand that information from other systems is not in the same “now.” Stuff may have changed by the time you see it. It is essential that business concepts accept these truths. This is no different from what took place in the years before telephones when people used handwritten messages and couriers to coordinate business.


By necessity, the solution to this temporary ambiguity (the uncertainty about the truth) must be handled in an application-specific way. Different business problems have different business mechanisms for coping with uncertainty.

One of my favorite notifications is: “Usually ships in 24 hours.” This tells you absolutely nothing about any guarantees, but it is incredibly useful—I am always adding or removing things from my shopping cart based on this ill-defined nonguarantee.


In large, loosely coupled distributed systems, we must also recognize that there’s more than one computer. Each of these computers will have its own time domain and its own sequence of transactional history. We must build our applications to cope with the ambiguity of these multiple histories.

What the Heck Are You Talking About?

When data is published in a document or sent in a message, there is usually an attempt to describe it so the reader can



Huge scale has implications on the consistency semantics of the data. It turns out that most large systems in the NoSQL (not only SQL) genre cannot support transactions across all of their data.



understand it. The nature of this schema for the data is intrinsically different in a very large and loosely connected distributed world. Here, I discuss some of the challenges in communicating what is supposed to be communicated.

Messages and schema. When a message is written in a distributed and heterogeneous system, the writing system (ideally) understands its semantics. At the time it is written, there is a context and intent. Hopefully, a schema *describing* the intention of the message accompanies the content explicitly or by reference. The goal of this schema would be to facilitate the understanding of the intent of the message.

Messages are immutable. They are written, sent, and possibly retried. It is considered buggy to have a message retry semantically different from the first attempt. Thus, when describing the message, the schema information should be immutable, too.

You know what you mean when you say it. When a message or document is written, the writing system knows its meaning. Frequently, some description or schema is attached to the document to facilitate the reader in figuring out what the sender meant. This description may be in any of a number of representations including XML (and its many schema variants). One nice thing about XML is that essentially all systems can parse it well enough to get confused about the semantics of the message rather than the syntax. This is a huge advancement.

I’m trying to understand you but your accent is thick. Unless the reader of a message or document is specifically programmed for it, there will likely be confusion. The meaning of the message, the interpretation of its fields, and much more will be subject to approximation and a loss of clarity. Different companies, different countries, and even different regions within a country have different understandings of data. Sometimes it is harder to communicate with a different department inside the same company than with some external partners. (Of course, there are always contextual issues that are notoriously difficult to capture. The schema author will likely do his or her very best to express the meaning of the data. But when modern readers try to decipher a medieval work, there are

cultural assumptions that may be as inscrutable as those posed to the parents of teenagers. Similar challenges pervade message and document schema.)

Who's the dog and who's the tail?

When two organizations try to communicate, there is always an economic dog and an economic tail. The dog wags the tail and the tail moves. In messaging and/or document definition, it is the economic dog that defines the semantics. If there is any ambiguity, the onus remains on the economic tail to work it out.


Walmart, for example, is a dominant force in retailing and dictates many things to manufacturers that want to sell through it. In addition to packaging and labeling standards, Walmart imposes messaging standards for communication with the manufacturers. Walmart prescribes the meaning, and the manufacturers adapt.

Schema versus name/value. Increasingly, schema definition is captured in the “name” of a name/value pair. This can be seen in the move from a SQL DDL (which is intended to be a tight and prescriptive statement of the meaning of the data) to XML (which is much more intended as the author's description of what was written in the message or document). Name/value pairs (and their hierarchical cousins in XML, JSON, and the like) are becoming the standards for data interchange.


We are *devolving* from schema to name/value pairs. The transition away from strict and formal typing is causing a loss of correctness. Bugs that would have been caught by a stricter description can now squeeze through.

On the other hand, we are *evolving* from tightly defined prescriptive schema to the more adaptable, flexible, and extensible name/value pairs. In very large, loosely coupled systems, adaptability and flexibility seem to offer more value than crispness and clarity.

Extensibility: Scribbling in the margins. Extensibility is the addition of stuff that was not specified in the schema. By definition, it is data the reader did not expect but the sender wanted to add, anyway. This is much like scribbling additional instructions in the margins of a paper form that was not designed for such additions. Sometimes the person reading the form will



When data is contained inside a database, it may be normalized and subjected to DDL schema transformations. When data is unlocked, it must be immutable.



notice these additional instructions, but sometimes not.

Stereotypes are in the eye of the beholder. A person dresses in a style usually intended to provide information to strangers. When people began to transition from living in small villages (where everyone knew you and knew what to expect from you) to living in large cities (where most of the folks you encounter are strangers), it became important to signal some information to others by dressing in a particular way.

People dynamically adapt and evolve their dress to identify their stereotype and community. Some groups change quickly to maintain elitism (for example, grunge); others change slowly to encourage conformity (for example, bankers).

Dynamic and loose typing allows for adaptability. Schema-less interoperability is *not* as crisp and correct as tightly defined schema. It presents more opportunities for confusion.

When interpreting a message or document, you must look for patterns and infer the role of the data. This works for humans when they examine a stranger's stereotype and style. It allows for flexibility for data sharing (which includes a cost for making mistakes).

Sure and certain knowledge of a person (or schema) has advantages. Scaling to an infinite number of friends (or schemas) isn't possible. The emerging adaptive schemas for data like stereotypes in people. While you can learn a lot quickly (but not perfectly), it scales to very large numbers of interactions.

Descriptive, not prescriptive schema. In very large and loosely coupled systems, we see descriptive, not prescriptive schema:

- *Descriptive schema.* At the time the data is written, the author describes what is intended.

- *Prescriptive schema.* The data is forced into a fixed format that is consistently shared by all authors.

The larger and more disconnected the system, the more impractical it is to maintain a prescriptive schema. Over time, the attempt at consistency becomes a fragility that breaks. Natural selection drives extremely large systems away from consistency and prescription.

Would You Like Some Mulligan Stew?

In the early 1900s, hobos living on the rails would frequently eat “Mulligan stew.”⁵ Members of the community would contribute whatever ingredients were available, and everything would be tossed into a community pot. Based on circumstances, there could be many different meats, fishes, vegetables, and other edibles all combined into the same stew. Leftovers would form the basis of tomorrow’s stew. While this dish can be delicious, it’s difficult to define its contents precisely.

Many large systems extract data from a tremendous number of sources. This disparate data is processed, crunched, shoehorned, and pounded into a mush—a Mulligan stew of sorts. By combining data and reasoning about its relationship to other data, astonishing businesses have sprouted such as Google and Amazon. This section examines some of the mechanisms by which this is accomplished and the compromises needed in the process.

Extract, transform, and load. Many data systems perform extract, transform, and load ETL: collecting data from one or more sources, converting it into a new form with a transformation, and then loading it into a data store for further operation (Figure 3). This is a valuable process growing in its use and scale. The transformation of the data is frequently a lossy operation, meaning that there is less information after the transformation than before. You can’t take the transformed data and go back to the original infor-

mation. It is much like creating a hamburger from some chunks of beef—you can’t go back to the original form (see Figure 3).

Lest you think this is meant as a criticism, I want to point out that I am very fond of hamburgers. Also, I am very fond of the gigantic data sets produced by these ETL operations, including the search indices of Google and Bing, as well as the product catalog from Amazon. While the result cannot support backward processing, which returns us to the source data, the resulting stew is really tasty.

Building a heterogeneous product catalog. Shopping sites such as Amazon, Google Shopping, and Bing Shopping gather data from literally millions of sources through a combination of data feeds from partners and crawling the Web. Rarely is this data in a standard format, and, if it is, there are always challenges with the semantics. (This same discussion can apply to creating search information for Internet searches.)

Typically, the first step involves cleanup. How can you unify the data so it can be compared and contrasted? For example:

► *Who’s the manufacturer?* HP, Hewlett-Packard, Hewlett/Packard, Compaq, Digital, DEC, H-P?

► *What’s the color?* Green, emerald, asparagus, chartreuse, olive, pear, shamrock?

Mapping into a single value increases the ability to see the data as equivalent, but you may lose knowledge.

Identity and ambiguity. There are

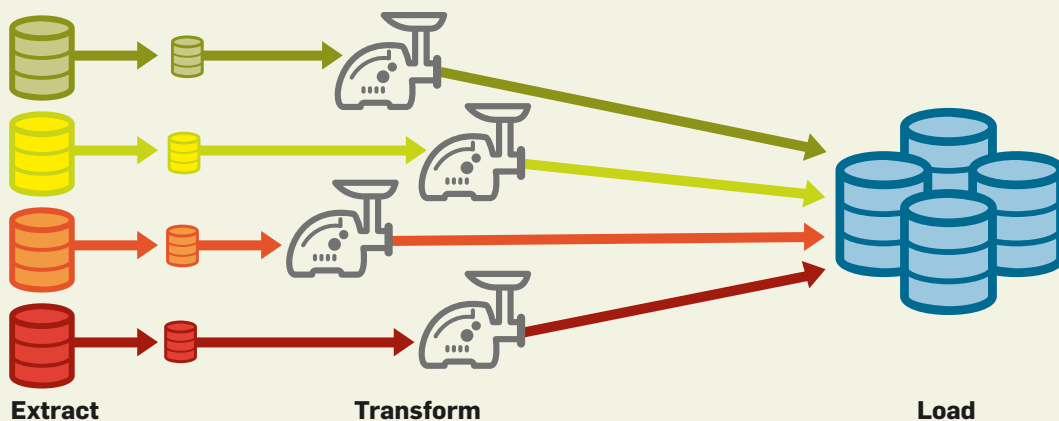
many competing and unifying mechanisms for assigning a unique number for a type of product—for example, GTIN (Global Trade Identification Number), UPC (Universal Product Code), and EAN (European Article Number). Amazon internally uses its ASIN (Amazon Standard Identification Number), which includes the book-centric standard ISBN (International Standard Book Number).

As an example of the challenges in standardizing identity, a different ISBN is assigned to each edition and variation (except reprinting) for each book. This means that there is a separate identity for the paperback and hardback of each book. Reviews and commentary on the literary value need to be connected separately across the identities. Similarly, some products have the same UPC even when they have different colors. For some reason, shoes carry no standard identification number but, instead, are identified by manufacturer and brand name (and some manufacturers recycle the brand names).

Based upon these tenuous identities, a product catalog must now try to discern which merchants’ offerings match other offerings. Can we determine an industry-standard identification? Can we accurately correlate this product from merchant A with the same product from merchant B so we can offer the customer a comparison?

Transformation and consolidation. Once multiple products from a set of merchants are recognized as the same and mapped to a unique identity, it is time to create a unified product de-

Figure 3. ETL operations can produce gigantic data sets.



scription. The best information from multiple merchants (and sometimes the manufacturer) is processed to create a catalog entry. Based on the identity, you want to get the best product description possible.

Observing patterns by inference. Yet another form of knowledge is gathered by patterns and inference. By studying the contents of data, it is sometimes possible to infer that two seemingly disparate identities are really the same. This happens when looking at a catalog of shoes from different merchants, for example. The lack of an industrywide identifier such as a UPC for shoes means they are particularly subject to erroneously interpreting two shoe descriptions as independent items.

Inference engines look at the relationships between identities (and attributes of the identities) to recognize that two ostensibly different items are the same. Once you realize two identities are the same, new relationships are recognized by merging the two identities into one. Inference engines are constantly appending new information to the data gleaned from the Web and partner companies.

Serendipity when you least expect it. The recognition gathered by inference engines is incredibly valuable. Some important areas of application include:

- *Fraud analysis.* The recognition of fraudulent patterns of credit card usage is vital in online credit card usage. It is only because of these fraud-detection mechanisms that online pay-

ments are economically viable.

- *Homeland Security.* There is tremendous traction in tracking surprising patterns of suspicious people. There is also a recent growth in *anonymizing* identities in a pattern that shares relationships without divulging identities and violating privacy.

- *Item matching in marketplace catalogs.* Are those two SKUs the same product for sale?

Inevitably, this “enhancement” of the source data results in new insight and increasing distance from the source data.

What kind of stew is that mulligan stew? Identifying the provenance of data is increasingly difficult. Where did this result come from? Who is responsible for the correctness of the data? When data from multiple sources is smashed together with data from other derived sources, what is the provenance? What is to be done when legal changes mean that some of your input data should no longer be used? The loss of licensing rights and/or the divestiture of a portion of your company can mean the input data is no longer yours to use. If that input data has contributed to the Mulligan stew of your intellectual property, you have a problem. (There is some irony to be found in complaining about provenance in an article that uses Wikipedia for some of its references. Hmmm...)

Through the looking glass. When considering the large and scalable applications that are gravitating to the NoSQL model, we must take into ac-

count the semantics of the derived data that inevitably accompanies them. The application probably wants a simplified and distilled view of the aggregated data. This view is inevitably a lossy prism that looks at a subset of the knowledge. But only by accepting the inevitability of the loss can we look for ways to gain business value from the resulting tasty stew.

Heisenberg was an Optimist

Werner Heisenberg⁶ showed that as things get small, you cannot know everything precisely. Well, it turns out, as things get big, we can get confused, too.

How certain are you of those search results? Web search results are very challenging. Web crawlers are, well, crawlers. Ensuring that search results are relevant to the person issuing the request is notoriously difficult because of the ambiguity of the request and the sheer magnitude of the Web. The demographics of the audience affect the desired results. Teenagers frequently want different results than their parents, for example. The requester’s location, interests, and recent searches dramatically affect the expected answers.

The U.S. census is hard. Any census of a large collection of people is very hard to achieve. You cannot accurately count people when you can’t be at every house at precisely the same moment. People move; people lie; people live with their girlfriends and don’t tell Mom and Dad. Should you count by address, Social Security number, name, or something else? What if someone dies after you have counted him or her? What if someone is born after the family’s house is counted but before the census is complete?

Chads and the election results. In the U.S. presidential election of 2000, the results ended up depending on the State of Florida. The vote in Florida was nearly tied, and each recount yielded a different answer. Much of the ambiguity came from the automatic voting machines and the presence of chads (illustrated in Figure 4) that occasionally made it difficult (and/or subjective) to interpret the voters’ intentions.

The vote in Florida was very close by any measure, with the first result showing a difference of about 0.01%.

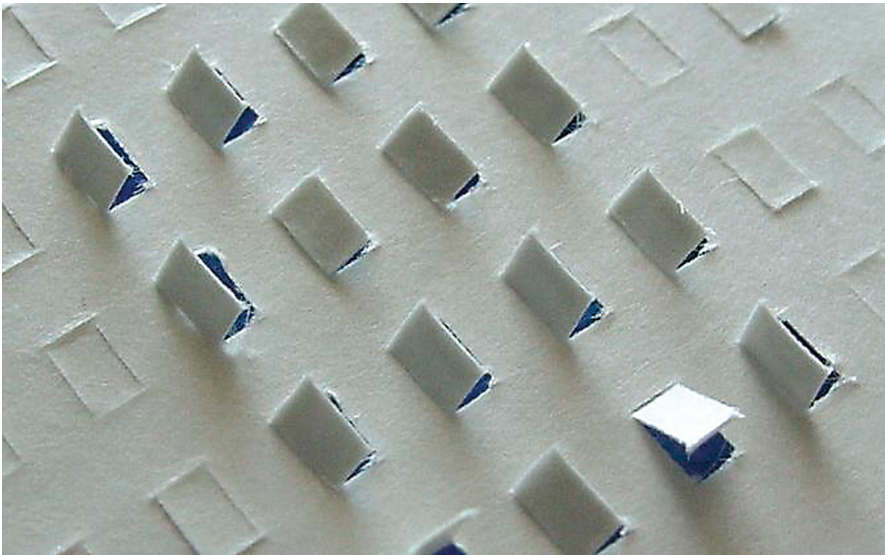


Figure 4. Chads created ambiguity in a large-scale data set—Florida’s vote count.

In any large system, the inaccuracies don't matter much until precision is required. If the winner has a 10% lead, there is no need for a recount.

Under scale, we lose precision. Big is hard! Time, meaning, mutual understanding, dependencies, staleness, and derivation all become a challenge. Heisenberg pointed out that at a small scale, uncertainties are a fact of life. In computing at a large scale, uncertainty is also a fact of life.

Conclusion

NoSQL systems are emerging because the world of data is changing. The size and heterogeneity of data means that the old guarantees simply cannot be met. Fortunately, we are learning how to meet the needs of business in ways outside of the old and classic database.

Data on the outside versus data on the inside. In a paper presented at the 2005 Conference on Innovative Data System Research (CIDR), I observed that data that is locked (and inside a database) is seminally different from data that is unlocked.² Unlocked data comes in clumps that have identity and versioning. When data is contained inside a database, it may be normalized and subjected to DDL schema transformations. When data is unlocked, it must be immutable (or have immutable versions).

Database normalization is designed to help avoid update anomalies. In large-scale systems, you don't update data, you add new data or create a new version.³ The treatment of data in large-scale systems is seminally different.

A new theory for data needed. The database industry has benefited immensely from the seminal work on data theory started in the 1970s. This work changed the world and continues to be very relevant, but it is apparent now that it captures only part of the problem.

We need a new theory and taxonomy of data that must include:

- *Identity and versions.* Unlocked data comes with identity and optional versions.

- *Derivation.* Which versions of which objects contributed to this knowledge? How was their schema interpreted? Changes to the source would drive a recalculation just as in Excel. If a legal reason means the source data may not be used, you


Another form of knowledge is gathered by patterns and inference. By studying the contents of data, it is sometimes possible to infer that two seemingly disparate identities are really the same.

should forget about using the knowledge derived from it.

- *Lossiness of the derivation.* Can we invent a bounding that describes the inaccuracies introduced by derived data? Is this a multidimensional inaccuracy? Can we differentiate loss from the inaccuracies caused by sheer size?

- *Attribution by pattern.* Just like a Mulligan stew, patterns can be derived from attributes that are derived from patterns (and so on). How can we bound taint from knowledge that we are not legally or ethically supposed to have?

- *Classic locked database data.* Let's not forget that any new theory and taxonomy of data should include the classic database as a piece of the larger puzzle.

It's a great time to be working with data. Lots of new and exciting things are happening! 

Related articles on queue.acm.org

A Call to Arms

Jim Gray, Mark Compton

<http://queue.acm.org/detail.cfm?id=1059805>

Exposing the ORM Cache

Michael Keith, Randy Stafford

<http://queue.acm.org/detail.cfm?id=1394141>

The Case Against Data Lock-in

Brian W Fitzpatrick, JJ Lueck

<http://queue.acm.org/detail.cfm?id=1868432>

References

1. Baker, J. Bond, C., Corbett, J. C., Furman, J.J., Khortin, A., Larson, J., Léon, J. M., Li, Y., Lloyd, A., Yushprakh, V. Megastore: providing scalable, highly available storage for interactive services. In *Proceedings of Conference on Innovative Data Systems Research* (Asilomar, CA, 2011); www.cidrdb.org/cidr2011/Papers/CIDR11_Paper32.pdf.
2. Helland, P. Data on the outside versus data on the inside. *CIDR 2005*; www.cidrdb.org/cidr2005/papers/P12.pdf.
3. Helland, P. Normalization is for sissies. Conference on Innovative Data Systems Research (2009); www-db.cs.wisc.edu/cidr/cidr2009/gong/20Helland.ppt.
4. Microsoft Developer Network. SQL Azure (2011); <http://msdn.microsoft.com/en-us/windowsazure/sqlazure/default.aspx>.
5. Wikipedia. Mulligan stew; [http://en.wikipedia.org/wiki/Mulligan_stew_\(food\)](http://en.wikipedia.org/wiki/Mulligan_stew_(food)).
6. Wikipedia. Werner Heisenberg; http://en.wikipedia.org/wiki/Werner_Heisenberg.

Pat Helland has been working in distributed systems, transaction processing, databases, and similar areas since 1978. For most of the 1980s, he was the chief architect of Tandem Computers' Transaction Monitoring Facility (TMF), which provided distributed transactions for the NonStop System. With the exception of a two-year stint at Amazon, Helland has worked at Microsoft Corporation since 1994 where he was the architect for Microsoft Transaction Server and SQL Service Broker. He is currently working on Cosmos, a distributed computation and storage system that provides back-end support for Bing.

© 2011 ACM 0001-0782/11/06 \$10.00