

Ontology: Things and Events.

Dr. Tom Johnston

MindfulData.com

Written sometime in 2006.

A Terminology of Things.

So we have the following terminology.

- **Things** are what there is.
- **Particulars** are types of things that exist in their own right.
- **Individuals** are instances of particulars.
- **Universals** are types of things whose instances exist as properties or relationships of things (either particulars, or other universals).
- **Concepts** are instances of universals.
- **Properties and relationships** are types of universals.

Events.

If things are what *exist*, events are what *happen*. Like things, events have both types and instances. But unlike things, we don't have convenient one-word terms to designate event types and event instances.

Events alter the properties and relationships of things. Things change by taking part in events. For example, a receipt event alters the on-hand balance property of a product at a store. The completion of an MBA degree alters the level of education property of an employee. The assignment of an employee to the position of store manager for a store alters the relationship between the employee and the store.

An event type is distinguished from all other event types by any or all of the following:

- the particulars that take part in the event;
- the properties and relationships of those individuals that are affected by the event;
and
- the properties and relationships of the event itself.

For example, the receipt of a shipment is an event. Particulars involved in the shipment might include the supplier, the carrier, and the department and company receiving the

shipment.¹ are One relevant property of the shipment itself is the items it includes, along with their product types and quantities. A relevant relationship of the shipment might be a relationship to the one or more purchase orders which authorized it. Properties of the event itself include when and where it took place, whether it was a profitable or unprofitable event for the supplier, or for shipping company, etc.²

So we have the following terminology.

- **Events** are what happens.
- **Types of event** are types of what happens.
- **Instances of events** (which the unqualified term “event” will more commonly be used to refer to) are points in time, or delimited stretches of time, at which an event of a designated type occurs.

Semantic Constraints.

There are four kinds of rules that suffice to preserve the semantics of data in databases, making that data an accurate representative of the things and events in the real world that it is about. Dr. Ted Codd first identified these four rules, and the history of the use of relational databases has confirmed his hypothesis.

1. The first is that objects must have an identity that distinguishes them from all other objects.
2. The second is that the properties of an object must be recognizable properties.
3. The third is that relationships among objects must relate objects that exist.
4. The fourth is that changes to an object must be valid, given the current state of that (and possibly of other) objects.

The first three rules (constraints), in relational database terms, are **entity integrity**, **domain integrity** and **referential integrity**. The fourth constraint can be called the **state-transition integrity** constraint.

¹ “Might include”, rather than “does include”, because the particulars and universals involved in events depends on what the company interested in those event wants to keep track of.

² “Include”, rather than “may include”, because although what is included or not is always a matter of what we are interested in, we almost never fail to be interested in when an event occurred.

Identifying, Describing and Relating.

These three functions should be done separately. This means that identifiers should not contain business data, because business data, by definition, is descriptive.

Identifying.

- This means that **surrogate keys** should be created for every token – for every row of every table in every database. These keys should be globally unique.
- No table should contain anything else in its primary key except a single attribute globally unique identifier. Dates should not be in primary keys. Purchase order numbers should not be in primary keys. No foreign keys should be in primary keys, including primary keys of associative tables.
- With identifiers based on business-meaningless surrogates, relationships are also freed of descriptive content. This is because relationships are implemented with foreign keys (in relational databases), and so will also be based on surrogates.
- Among descriptive, non-key attributes of data, there are three categories. One is the set of attributes defining the **business key** for a table. This is the set of business-meaningful data that makes rows in a table unique. Normalization must be defined against business keys; it's meaningless when defined against surrogate keys.

Describing.

- Next, the required attributes (including foreign keys) must be defined.
- Finally, the optional attributes (including foreign keys) must be defined. However, optional attributes would often not be optional if subtypes were defined, and the attributes migrated down to one of the subtypes. It is optional because there is one subtype where it is applicable (and required), and another subtype where it is not applicable.
- This subtyping strategy takes care of the optional (nullable) case based on applicable or inapplicable. However, some attributes may remain nullable even after this migration, because for them nullable means “applicable, but currently not known”.
- Part of the process of transforming data from local to enterprise in status is to put it in a **canonical form** in which the attributes used to identify and relate contain no semantics other than those roles, i.e. are surrogate keys. At the same time, **normalization** should be applied, and based on the business data that constitutes unique identity.

Relating.

Ontologies and Data Models.

- An **ontology** consists of the types of things relevant to a subject area, and the rules governing the relationships among those types of things.
- So a data model records an ontology, with the tables in the model representing the types, and their relationships representing the relationships among the types of things.
- But two or more data models, with different tables and relationships, can both fully represent the same subject area. Both can be fully normalized. What, then, makes one better than another?
- One important differentiator is how well the models express the ontology of the subject area. An “**ontological correct**” **model** is one whose tables are one-for-one with the types of things in the subject area, and whose relationships are one-for-one with the relationships among those things. Few models are completely correct, ontologically. But some are much more correct than others.
- An ontologically correct model is more intuitive, and more clearly conveys what it “is about”. This makes the code and queries written against it correspondingly clear. This clarity results in fewer bugs, and fewer confusions.
- Often, the more ontologically correct model is also more expressive, supporting queries that the less correct model cannot. Often, we are not even aware of these deficiencies in the poorer model until the better one is created, and a comparison is made.

Events and Relationships: Transactions and Balances.

- A **transaction** is the record of an event. Some events we don't bother recording in our database. These are changes to things, where there is no table of the transactions that caused the change. Updating an employee's date of birth, for example.
- Some events we do record in our databases. Transaction tables are the records of those events. Each transaction represents one event, like a receipt, a sale, a transfer, a hiring, a promotion, etc.
- If events are recorded, it should be possible to reach the current state of a thing by going back to a previous state, and applying all the transactions from that point in time to now. If only some transactions are kept (the important ones), then this will not be possible.

- **Transactions** update balances. **Balances** are the net of transactions. Balances are the record of a relationship between things. Current balances, then, are the current status of the relationship. For example, a credit card balance is the record of a relationship between you and the bank that issued the card.
- In the ontology, transactions are events, and balances are relationships among things.
- Often, these relationships (balances) exist in hierarchies. A chart of accounts is a hierarchy of balances.

Facts and Dimensions.

- **Facts** are what is true as of a point in time. Transactions and balances are both facts. These facts are the ones most often used as facts in the star-schema, dimensional data mart sense. This is because dimensional modeling focuses on the activities of a business. Transactions are records of those activities, and balances are the point-in-time relationship states resulting from those activities.
- Other facts are not considered facts in the dimensional modeling sense. For example, the end-of-month snapshot of employees, over say two years, is a set of facts because each is a point in time view of a thing. But in this section, we will use the term “fact” only in the dimensional modeling sense.
- **Dimensions** are perspectives on facts. They are categories for facts. For example, store, sku and date are dimensions for inventory balances, and for inventory transaction totals, such as receipts, sales, transfers, returns and adjustments.
- Dimensions usually have a hierarchical structure. Facts are attached at the “leaf nodes” of their dimensions. This permits us to “**drill-down**” from the top of one or more dimensions through to the bottom, and then down to the transactions. It also permits counts, sums, totals and averages to be “**rolled-up**” from the transactions to the various levels of individual dimensions, and of combinations of dimensions.
- Often, summary-level metrics involve too much roll-up to be computed “on the fly”. So summary tables, called aggregate tables, must be physically created and maintained. But given the number of levels of several dimensions, and the number of members at each level, the storage requirements and processing time to create an aggregate for all the combinations of all the instances of these dimensions, is usually prohibitive. So only some levels and combinations of levels can be materialized.

Types of Tables.

First a word about types. Types depend on a context. For example, from the point of view of relationship dependencies among relational tables, all tables belong to one of only *three* types, those being:

- **Kernel tables**, ones whose unique identifier does not contain any foreign keys. Policy Holder and Invoice for example. Rows of kernel tables do not depend on any relationships in order to distinguish themselves one from the other.
- **Dependent tables**, ones whose unique identifier contains a foreign key back to a kernel table, together with a unique identifier for each instance of the dependent. A reference to a row in another table is a necessary component of the set of attributes which distinguish these rows from one another. Covered Family Member (of Policy Holder) and Line Item (of Invoice) for example.
- **Associative tables**, ones whose primary key contains two or more foreign keys, and no non-foreign key components. References to rows in the tables related by the associative table are sufficient to distinguish rows of the associative table from one another.³ Customer / Salesperson and Invoice / Payment for example.

The context in which kernel, dependent and associative tables are types has to do with relationships among tables, and whether they are necessary and sufficient (associative tables), necessary but not sufficient (dependent tables), or neither necessary nor sufficient (kernel tables).⁴

So within what context do I distinguish different types of tables? That context is an *ontology*. “Ontology”, of course, is a *big* word, one that I will discuss in depth elsewhere. For now, think of it as a set of basic categories of what there is. The formal study of ontology goes back to Aristotle, in particular to his works *Categories* and *de Interpretatione*.

³ The definitions would have to be extended a little to be truly exhaustive of all cases. What about, for example, a table with three foreign keys, or perhaps two foreign keys plus a date, as the primary key?

⁴ The fourth case – that in which relationships are sufficient but not necessary – is ruled out because, for reasons I will not go into here, every component of a relational table’s unique identifier must be necessary.

Individuals and Particulars, Concepts and Universals.

Tom Johnston
September, 2005

Now let's see how the ontological categories defined in Part 1 are reflected in different types of database tables.

Particulars, Individuals and Persistent Object Tables.

Persistent object tables are tables which represent particulars. Each of their rows represents an individual. Some typical examples include:

- Customers
- Salespersons
- Invoices
- Invoice Line Items
- Organizations
- Zip Codes

Some examples of tables which do not represent particulars include:

- Accounts Receivable
- Customer Status Codes
- A Customer Salesperson cross-reference (associative) table.

At the end of this essay, it should be clear why the latter three tables are not persistent object tables. In case it isn't, I'll briefly explain why.

Universals, Concepts and Reference Tables.

Reference tables are another type of table. They are sometimes called "lookup tables", or "code tables" or even "domain tables". In general, each row of a reference table has a code and a description of what it means. Sometimes it has an abbreviation instead of a code, and a full name instead of a description.

Color, and its instances red, blue and green, would be represented in a database as a reference table with three rows. Customer status code, and its instances VIP, in good standing, standard, poor and high risk, would be represented in a database as a reference table with five rows.

Instances of reference tables exist as properties (or relationships) of instances of things, either of individuals (instances of particulars) or of concepts (instances of universals).⁵

⁵ Note that an individual can exist only once, while a concept can exist as many times as there are things that "instantiate" it. This is another important difference between particulars and universals.

In contrast to particulars, universals don't exist through time, and so cannot change. Color, for example, doesn't exist through time, although there certainly were colored things in the past, are colored things today, and will be colored things in the future. But things exist in time in the sense that they exist in their own right, and except for eternal things (if there are any), their instances begin to exist at a point in time, and cease to exist at a later point in time. This isn't true of the instances of color, or of customer status, or of any other universals.

Here's another way to look at the unchangeability of universals. Certainly the words we use to designate universals often change their meaning over time. "Naughtiness", in the King James bible, meant "evil". These days, it means something much more frivolous. But if we are willing to distinguish between universals and their representation in language, we can say that the universals did not change, but our labels for them did.

So suppose we have a reference table of Customer Status Codes, with one row in that table being the code "VIP". And let's say that the business decides, on a certain date, to change the definition of "VIP" customer from "customer who has made over \$10,000 of invoiced purchases in the last twelve months" to "customer who has made over \$10,000 of invoiced purchases in the last twelve months and whose credit score is above 700". Hasn't the concept of VIP customer changed?

We could indeed say that. And if we did, we would be using the term "change" in a wider sense than I have been using it. There is nothing necessarily wrong with doing that. But I am creating a set of distinctions by using "change" to mean "change in one or more properties or relationships of particulars".

This, of course, makes a tautology of the statement that universals do not change. My justification for using "change" in the sense I am using it is that it helps me explain my ontological categories in a way, hopefully, which is clear enough to allow others to use those categories in a way consistent with my intended use for them. Another justification is that I think that if we did extend the term "change" to include change in universals, we would have to admit that, in a sense admittedly difficult to explain, these are two different senses of the term "change". In other words, I think that my use of the term "cuts nature at the joints", i.e. is somehow natural rather than artificial.

Event Types, Events and Transaction Tables.

Transaction tables are a third type of table. Each transaction in a transaction table represents one event. The table indicates the type of event in question.

For example, sales are events. This kind of event usually has four main participants – buyer, seller, what was sold, and what was given in exchange. In non-barter sales transactions, goods or services are sold, and what is given in exchange is money, or the

promise to pay money at a later date (i.e. credit). All four participants in this type of event are particulars. All four participants in each instance of this type of event are instances of those particulars, i.e. individuals.

Unlike particulars, but like universals, events can't change because they don't endure through time. Events *happen*, and then they are over. For this reason, the only reason to update a transaction – the representation in a database table of an event – is to correct a mistake. A transaction is never updated because the event it represents changed. Again, events don't change. They happen, and then they are over with.

Particulars, by contrast, do change. So we might update a row in a Customer table, for example, because the customer represented by that row changed his address. We might also update a row in a Customer table because the original data entry was in error. But for transactions, only the latter reason can justify an update.⁶

Transactions may involve only one thing. For example, hospital staff may take a patient's temperature twice a day, and record it on a chart. The chart is a transaction table, and each entry is a transaction. But the patient is the only thing affected by the transaction. In these cases, transactions record the change of a property of a thing, in this case the temperature of a patient.

But many transactions involve two or more things, and what they reflect is a change in the relationship among those things. These kinds of transactions have a cumulative effect on that relationship, and the current state of that cumulative effect is recorded as a balance.

A Special Relationship: Balance Tables.

An inventory receipt, for example, increases the on-hand quantity for the type of product received. It also increases the total accounts payable for the receiving company. It is the equivalent of a sale, only one in which the company is the customer instead of the seller.

The transactions that most businesses are interested in are those that affect relationships that have quantitative measures. A payment is received, an event. The relationship between the payer and payee is altered by the amount of the payment. That relationship is recorded in a revolving credit balance record, or else an accounts receivable balance record. The payment is recorded as a credit, and the balance due is decreased by that amount.

⁶ Note that I talk about updating a transaction, not about changing it. I do this because I am trying to draw parallels between ontology and databases. So to keep things clear, “change” will be an ontological term, referring to what happens in the world, while “update” will be a database term, referring to what happens in a database as a reflection of what happens in the world.

These records are called balance records because, like a see-saw, the relationship they affect swings now one way, now back the other way. Each change to the relationship is triggered by a transaction, and the net effect of those changes is the current balance of the relationship. The transactions are so important, however, that we keep track of them, and usually provide the ability to “drill down” into those transactions to get to a history of how the current state of the relationship came about.

Balances aren't the only kind of relationship. There are other kinds. A Customer / Salesperson Cross-Reference table – an associative table, in relational terms – represents a relationship between customers and salespersons. This table might represent a business rule which states which salespersons are authorized to sell to which customers. This table can be updated by a transaction, but those updates are not important enough that we keep track of those transactions, or provide the ability to “drill down” into those transactions from the relationship.

To summarize: companies are all about ongoing relationships. Those relationships are affected by events, which are recorded as transactions. Rather than adding up all transactions that affect a relationship each time we want to see the current state of that relationship, we keep a current state of the relationship record – a row in a balance table. Financial account tables are balance tables; each account number uniquely identifies a particular relationship, and the metrical properties of that account tell us the current status of the relationship.

Four Types of Tables.

So derived top-down from my background in the philosophical study of ontology, and derived bottom-up from my decades' long experience with business databases, I have aligned one with the other to come up with the following types of tables, and their correlation with ontological categories.

- **Persistent object tables.** These tables represent particulars. Their rows represent individuals. The principle of the identify of indiscernibles, and the distinction between essential and accidental properties, are reflected in the entity integrity constraint, that no two rows of such tables are identical, either in total, or in their primary key columns.
- **Reference tables.** These tables represent universals. Their rows represent concepts. The identity of indiscernibles and the essential vs. accidental property distinction apply here as well. If we think of reference tables as consisting of a code and a description for that code, the code is the essential property. But a change in the description, the non-essential property, is not a change which leaves the concept alone. It is a change which introduces a new concept that replaces the old one by “taking over” the code.

- **Transaction tables.** These tables represent types of event. Their rows represent specific events of that type. Events don't change. They happen, and then they are done with. Because of this, transactions are updated only if they were entered in error. They are never updated because what they describe has changed. Again, events don't change.
- **Balance tables.** These tables represent relationships whose state is altered by a series of transactions and which are important enough to be persisted. Practical considerations aside, any balance could be recreated, as needed, "on the fly", by summing up all the transactions that affected that relationship. But balance tables describe the web of ever-changing relationships that define the internal and external state of the company as it has been affected by that company's internal and external interactions with other things.

Ontology: Does It Matter?

What I'm trying to do is bring a high-level pattern, from another discipline (in this case, ontology), to bear on the practice of designing sets of tables to efficiently and effectively record and return information. Because this pattern is so high-level in comparison to collections of tables in individual databases, the value is not easy to see.

One value is that it is a repeating pattern that we can use to "chunk" our experience. But from this perspective, it is no better (or worse) than the pattern of three types of tables we mentioned above or, more generally, no better or worse than any other applicable pattern.

But I think it's a valuable pattern in a much stronger sense than this. And I will talk about this additional value in a later essay on ontology, the semantic web, and the IEEE's 1600.1 Standard Upper Ontology workgroup.

This set of ontological categories is, as far as I know, unique. So also is this set of four types of database tables.

Wrap-Up.

In the language of ontology: particulars endure. They have properties, and relationships. Events happen, and in happening they affect particulars, both their properties and their relationships. Ongoing relationships often have a current state that is continually being affected by certain kinds of events. For those relationships, we often want to keep track of and review the events that impacted them.

In the language of databases: some tables represent persistent objects. The foreign keys those tables have represent their relationships, and the non-foreign key columns represent their attributes. Transactions take place which update the values of certain attributes. Other transactions take place which update the state of certain relationships. We generally don't keep transaction tables for the former kind of transactions. But for the

latter kind of transactions, when the relationships they affect are important enough to be represented by an account, we want to know the current state of the account – its balance – and often want to be able to review the succession of transactions that generated that balance.

The first description is a basic ontology. The second is a discussion of types of database tables. The parallels are important and valuable, although that is a claim I have yet to defend.

And, finally, to why each of the tables in our short list of non-persistent object tables, is a non-persistent object table.

An Accounts Receivable table is not a persistent object table because it is a balance table. As such, it represents a relationship, not a particular. Accounts Receivable accounts do not exist in their own right; they exist only as a record of the current state of a relationship.

Customer Status Codes are reference tables. They represent universals, not particulars. Their rows represent concepts, not individuals.

A Customer Salesperson Cross-Reference (associative) table is also a relationship table, albeit not a balance table.

A rule is a constraint on an object. It's called a rule because the term "business rule" is popular, and that term is understood to have something to do with the meaning of data. A business rule is a constraint on data that wholly or partially gives that data its meaning.

Can relationships have properties? Can properties be related? Can relationships be related?