Chapter 8
Data Integration: Principles and Practice

Mark Schildhauer

Abstract Data integration is the process of combining (also called "merging" or "joining") data together to create a single unified data object from what were multiple, distinct data objects. The motivation for integrating data is usually to bring together the information needed to jointly analyze or model some phenomena. By producing a single, consistently structured object through data integration, the process of further manipulating those data is vastly simplified, while presumed relationships among the data are clarified.

Data integration is essential for many scientific disciplines, but especially in disciplines such as ecology and the environmental sciences, where processes and patterns of interest often emerge from interactions among numerous complex physical phenomena. Observations of these distinct phenomena are often collected by disparate parties in uncoordinated ways, using different data systems. It is then necessary to gather these data together and appropriately integrate them, to clarify through further modeling and analysis the nature and strength of any relationships among them. Synthesis studies, in particular, often require finding, and then bringing together disparate data in order to integrate them, and reveal new insights.

This chapter describes aspects of data that are critical for determining whether and how data can be integrated, and discusses some of the theoretical considerations and common mechanisms for integrating data.

8.1 Introduction

Data integration is the process of combining, merging, or joining data together, in order to make what were distinct, multiple data objects, into a single, unified data object. Data integration is one of the most fundamental operations that researchers and analysts typically must master, as many interesting scientific questions can be investigated only if the data needed to address them are assembled together. A typical motivation for data integration is to bring together data of similar or

M. Schildhauer (✉)
National Center for Ecological Analysis and Synthesis, University of California, Santa Barbara, CA, USA
e-mail: schild@nceas.ucsb.edu

F. Recknagel, W.K. Michener (eds.), *Ecological Informatics*,
DOI 10.1007/978-3-319-59928-1_8

complementary kinds, in order to better inform models and analyses about phenomena of interest. Data are also integrated in order to discover or test whether potential relationships exist among the pieces, or to expand the spatial, temporal, and thematic ranges over which scientists can explore potential relationships.

There are also many situations in scientific research where experiments are carried out in a controlled environment by manipulating a well-defined set of "variables" so that variability among unmeasured features is considered to be unimportant relative to the outcome of interest. Such experimental studies provide a strong inferential basis for testing and refining hypotheses, particularly when the processes of interest are driven by relatively few factors in strongly determinate ways. Many of the advances in molecular biology over the past decades, for example, have been gained through carefully controlled experiments based on sets of mutually exclusive hypotheses (Platt 1964). The data supporting these types of research is often fully self-contained with minimal need for further integration with other data.

While highly controlled experiments such as those commonly found in laboratory work can advance scientific understanding in powerful ways, studies of the natural environment are often not amenable to such designs as many factors—climate, topography, hydrology, soil composition, or the abundance and identity of surrounding organisms (including humans!)—cannot all be simultaneously manipulated or controlled. Yet these factors may be significant in structuring processes at the ecological population, community, and ecosystems levels. In essence, ecological systems can be highly complex, involving extensive interactions and feedbacks among many potentially critical factors. This leads to a dilemma as to how scientists can collect data on all these potentially critical factors, given the relatively limited resources that individual researchers, and even research teams or projects, typically have available to record observations of *all the potential factors* that could influence the patterns and processes shaping the environment at multiple scales.

Many field sciences (ecology, geology, oceanography, etc.) especially benefit from having additional data available from the particular spatiotemporal region where their observations were acquired: one might combine lists of bird species collected by different individuals from the same area to gain a more comprehensive record of the local avifauna; or one might merge data on tree growth rates with data sets about air temperature, soil type, and precipitation, that were collected from the same geospatial region over the same time period.

The need for data integration is thus pervasive throughout much of the environmental and earth sciences. Indeed, data integration is one of the principal activities involved in doing synthetic analyses. Much of the success of ecological synthesis studies over the past few decades can be attributed to arduous but fruitful efforts that brought together "existing data" to generate novel insights, activities that were often facilitated by synthesis centers that support knowledge and data integration activities (Carpenter et al. 2009). Regardless of the motivation, however, data integration requires that researchers understand structural and semantic aspects of

the data, as well gain some mastery of the mechanics involved in transforming and manipulating data to produce appropriate and re-usable integrated data products.

In this chapter we describe the basic concepts and mechanisms of data integration. As there are many types of specialized data formats and these can require special considerations based on the nature of the phenomena that the data "represent", we focus here on understanding the structure and mechanics of dealing with *tabular data*. While we will not cover the technical details of dealing with other common data types conventionally stored in other formats—e.g., gene sequence data, or raster or vector formatted geospatial data—many of the conceptual and operational aspects of working with data described here pertain to those formats as well.

8.2 Essential Characteristics of All Data

Data integration can take many forms depending on the characteristics of the data to be integrated. There are three essential characteristics of all data, however, that should always be considered by a researcher when planning for data collection, as well as for strategizing how to integrate data. These characteristics are the **semantics** (or *meaning*) of the data, the **structure** (*form* or *format* in which the data are documented or stored) of the data, and finally the choice of **syntax** (*specific programming language* or *application*) used to define, store, query, and retrieve the data.

Many scientists take for granted the meaning and structure of their data because they are intimately acquainted with the theories and methods for investigating their phenomena of interest, the conventional terms used to describe those phenomena, as well as various specific tools for acquiring, storing, and manipulating data about those phenomena. However, this can lead to highly idiosyncratic modeling of data, structured in ways that are only comprehensible to the data creator, and lacking in documentation that would enable others to understand the content or structure of the data. This lack of standardization in terminology and modeling is increasingly problematic in an era where the benefits of preserving and documenting data for interpretation and re-use by others is becoming more and more critical (see Cook et al. 2017). Data can serve several purposes, especially in the environmental and earth sciences: as records of the state of diverse natural phenomena at various times and places; as the empirical basis supporting important research findings and hence "reproducible science"; and as source information having second or third "lives" when re-purposed and integrated into synthesis research activities (Hampton et al. 2015). For these reasons, scientists are increasingly taking greater care in planning the structure of their data, and documenting data contents as well, using available tutorials and published best practices (Michener 2015; Strasser et al. 2014).

The concept of metadata—explored in depth in Chap. 5—was alien and confusing to many researchers as recently as a decade ago. As of 2017, however, most scientists are now aware of the importance of providing critical additional information (*metadata*: "higher order, beyond" data) about their data, if it was not already explicated within their original data structure and documentation. With the continuing growth of Internet technologies and computational power, the potential to integrate and jointly analyze data of extremely high volumes or dimensions has never been greater, and is catalyzing a growing culture of collaboration, synthesis, and data-sharing throughout many of the earth, environmental, and social sciences. But data integration can only proceed with ease and confidence if the data are well described and well structured. This means that adequate documentation or metadata in some form, is critical—describing the ***semantic content*** of the data (what the data are about); and that the data are organized using standard ***structures*** (how they are stored), rather than idiosyncratic, custom, or inappropriate ones. Following best practices in defining and structuring one's data will make them discoverable, accessible and amenable for further integration and analyses *by computers*, rather than arduous, manual methods such as copying and pasting data across worksheets. Finally, the choice of ***syntax*** by which the data are created and manipulated/integrated, is important. Although there may be many variations of computer language syntax for defining and describing data, there are also many commonalities that we outline in this chapter.

Note that every analytical framework approaches data integration using its own syntax and, to some extent, with specialized conceptual models of the data as well. In the case of tabular data, however, these all conform to a great degree to the *relational model* of data developed by Edgar Codd in 1969, with syntactical representation for creating and modifying relational data through Structured Query Language, or SQL (pronounced as "S-Q-L" or "sequel") (Codd 2000; also see https://en.wikipedia.org/wiki/Relational_model; https://en.wikipedia.org/wiki/SQL). As SQL is also an ANSI and ISO standard, we primarily frame our data integration examples using SQL syntax. Data analysis software such as "R", Matlab, and SAS have similar but often more idiosyncratic and less standard terminologies than SQL for describing data integration operations, depending on which specific packages, modules, or libraries of those frameworks one uses. However, there are often close analogues to SQL syntax for manipulating data in each of these frameworks. For these reasons, understanding some basic SQL and the relational data model is very useful to any analyst who will be doing lots of data creation, manipulation, or integration.

8.3 Data as Records About Reality

Scientists today use many sensors and instruments to collect their data, but in the not too distant past, individuals mainly used their own keen senses—visual, auditory, tactile—to describe and "measure" aspects of nature, and to record these as

observations. Of course, humans' direct sensory capabilities are highly limited (witness a dog's sense of hearing or smell relative to typical human beings), then filtered through our brains, where further interpretation of what was observed (whether real or imaginary) is (cognitively) conceptualized, and recollected or recorded as "data". Our everyday experience is largely based on such interpretations of sensory-based inputs about physical reality, whether scientifically informed or not. Scientists are specifically trained to be acutely aware or knowledgeable about various aspects of this reality, and typically describe and measure aspects of nature with greater rigor, striving to be more deeply perceptive, objective, and unbiased, compared with a naive observer. While it is debatable whether any observations can be fully objective or unbiased, one can point to science as an empirically-based "way of knowing" that has had unprecedented success in predicting and informing many aspects of the behavior of physical reality, compared with other methods, such as "blind faith" (not the band).

When data are used to support details about some natural event or occurrence, we also call such data "evidence". At this level, the very notion of identifying objects and processes in the natural world, and recording their characteristics and interactions with other objects and processes, touches on the domain of philosophy of science (Quine 1981; Taper and Lele 2004). We routinely draw distinctions among objects in the world, ascribe various characteristics to them, and then group these objects into sets and types (e.g., biomes, chairs, fish, and clouds), based on instances (or individuals) that actually occur and that we have measured or observed. For example, there is a notion of a chair, and there is also the instance of chair that you may be sitting in as you read this. We create a notion of "chair" based on some functional characteristics to which we then attribute membership to individual instances. Is a "table" a "chair" when someone is sitting on it? These issues are brain-teasers, and are discussed largely in the realm of philosophy, specifically, the branch of metaphysics called "ontology"—that inquires as to the nature of reality or being. Nevertheless, such issues have relevance to science, for science and empiricism involve describing, measuring, and classifying ("typing") events, occurrences, or physical objects in the material world and their interrelationships, so it is important to be thoughtful and critical about the basis for our understanding.

Most scientists probably subscribe to the perspective of "naturalistic realism" or "scientific realism", and do not worry much about the underlying basis in "reality" of the concepts and entities that they measure and describe (Hempel 1970), but there are reasons to be cautious about too much naiveté relative to the ontological status of all scientific phenomena (Kuhn 1996). Some reading about and reflection upon these issues may challenge scientists' confidence about the concreteness of their observations, and provoke greater reflection on the nature of the phenomena they are studying, as well as how they are "documenting" these phenomena as data. Fortunately, scientific conceptualizations of reality are continually challenged and amended by empirical observation and experiment, to the point where today we acknowledge that even our common day-to-day experience of space and time does not necessarily conform to some deeper reality, e.g., quantum entanglement (Greene 2005).

8.4 Record-Keeping and Prose Documents as Data Integration Challenges

Until the digital age, much "data" of scientific interest was collected in the form of notes, diaries and sketchbooks. The Codex of Leonardo da Vinci is an outstanding example of this form of scientific recordkeeping. These record-keeping formats typically involve lots of "natural language" descriptions (e.g., in prose English or Chinese)—but also include "richer media" such as illustrations or digital images with captions/descriptions, rough mockups of graphical trends (sometimes looking quite nice by using modern graphics software), and even areas of "well-structured data"—small tables or lists containing observations. These mechanisms for record-keeping are still useful for many field biologists, due to the expressiveness of natural language descriptions. Modern day field notes might also include descriptions of the human or environmental context surrounding the observations (metadata), along with imagery, additional essential metadata (location, date, observer, etc.) and other pertinent information as "annotations" (Canfield 2011).

Indeed, many great scientific insights from "data integration" in the past did not involve the assistance of computers running analyses on well-structured data, but rather from "data" collected and stored in the form of these lab or field notebooks. The theory of natural selection, for example, emerged from Darwin's integration of patterns perceived in nature, documented in field notebooks filled with observations of plants and animals, and their interactions and relationships (Fig. 8.1).

Today we can also consider these "natural language" documents as types of data that can be structured and integrated into a "corpus" which can be further mined or analyzed through "Natural Language Processing" (or NLP). The ability to rapidly automate the acquisition of large numbers of relevant text documents by scouring the Internet, and then organizing and "mining" these for patterns and relationships,

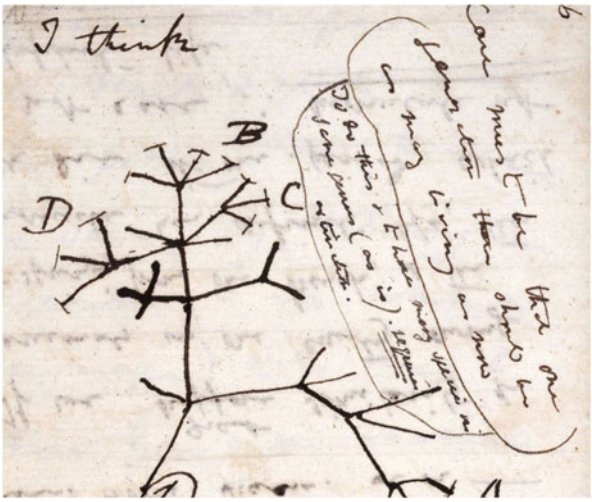

Fig. 8.1 Page from Charles Darwin's "First Notebook on 'Transmutation of Species'" (1837)

is a relatively recent capability enabled by modern computational power. NLP is enabled by "Big Data" analysis and involves "integrating" text documents, but will not be touched upon further here (Norvig 2009).

8.5 Formal Data Structures Facilitate Integration

Several formalized data structures have existed for millennia. Financial ledgers recording barter/trade transactions were used in ancient Mesopotamia and various highly structured representations of the movements of the sun or moon, viz. calendars, have existed since the Bronze Age. For modern scientific data, however, several common, well-structured data formats exist. The specific implementations and interpretations of these data structures can vary somewhat, depending on what programming languages or analytical tools one is using. Aside from the potentially subtle but significant differences in the semantics of data structures, implementations can also vary relative to the syntaxes (language commands) required for their construction and manipulation. For example, the specific meanings of the following data structures and how they are created or manipulated vary among Python, R, and MATLAB. Nevertheless, these variations are mainly in the fine details and the following common usages of these named structures (Sects. 8.5.1–8.5.4) will generally be accurate.

8.5.1 Sets and Sequences

Sets and sequences, including lists, vectors, and linear arrays—are (typically) one dimensional data structures consisting of individual *values* (also called *elements*) that can be strictly ordered or not. While any collection of *unique* elements sharing some common property can constitute a *set*, elements in a *sequence* can be *repeated* and referenced by mapping to an *index* position within the set. This is necessary in order to readily refer to elements (or subsets) within the sequence, without having to specify their actual value. By convention, in many programming languages, square brackets [. . .] or braces {. . .} or even parentheses (. . .) are used to delimit vectors, arrays, and lists. Examples of lists are:

- items one must remember to purchase [wine, cheese, crackers, plates, cups]
- types of butterflies seen on a hike [*Danaus plexippus*, *Morpho menelaus*]
- the months in a year
- the height (m) of a plant measured on the first of each month from 2003 to 2015 [0.45, 1.18, 2.46, 5.78, 6.38, 7.72]

If the first list above was called "picnic", then picnic[2] would equal "cheese" (the second element in the list); although in many computer languages, the origin of indexing starts at "0" not "1", so the second element would be: picnic[1]. In vectors all elements must be of the same type, while in lists, elements can be of mixed types, e.g., both character and numeric.

8.5.2 Matrices

Matrices are two dimensional, rectangular arrays of numbers or expressions, where in a formal mathematical sense, index-ordering of the values is fixed. The two-dimensional or rectangular matrix is common, and has a formal structure such that the exact ordering of values is fixed across both the first (often called "row") and second (often called "column") dimensions. Individual elements are identified and accessible via an index, e.g., X [i, j], where X [i+1, j+2] has a specific relationship to all the other matrix elements—referencing the cell one row "below", and one column to the "right". Thus, matrices cannot be arbitrarily re-sorted by row or column without changing the nature of the matrix (unlike a table, as described below). Matrices also needn't be solely numeric, though some language/software tools require that. Finally, matrices are often confused with, or somewhat misleadingly used to describe structures for displaying *cross-classified data*.

8.5.3 Cross-classifications

Cross-classifications are effective ways of *presenting* data when data elements are described according to two attributes, e.g., *seed shape* AND *seed color*; or *site* AND *species*, where the cell values could contain counts or other measures like total weight, etc. (Fig. 8.2). Cross-classifications are not, however, the best way to store and update raw data. One can see from the example in Fig. 8.2, that adding a new category of color or shape, such as "green-yellow", or "dappled", might require adding another column or row of data, and possibly re-classifying existing instances into these new categories. The *Table* structure described next provides a much better way for storing these types of data with great flexibility in splitting and lumping categories, and transformation to a Cross-classification format easily achieved a few simple transformation rules.

8.5.4 Tables

Tables are extremely common data structures used extensively in many natural and social sciences. There is both a common vernacular use of the term to indicate any structured "rectangular" data object, as well as a more formal definition that is grounded in the mathematics of set theory and relational algebra (Codd 1970).

In both cases, a table (equivalently called a "*relation*"), can be envisioned as a set of rows and columns, in which each column contains some particular type of

Fig. 8.2 Cross-classification data structure

SHAPE / COLOR	Green	Yellow
Smooth	323	46
Wrinkled	59	11

measurement or variable (also more formally called "attributes"), while the rows represent individual records or observations consisting of values for each of the column measurements. Note that these values can be "missing" or blank for any number of the columns in a row and any number of rows in the table. It can be a bit confusing to describe the structure of tables, as there is lots of variability in the terminologies that are commonly used (Fig. 8.3).

Tables are typically constructed such that rows collectively represent a set of records about some single "entity type"—where an "entity" represents some thing or process that has features that are measurable or named. Thus, a table might describe entity type=*People*, and measured fields might include a surname and birthdate. Rows would each represent an "instance" of that entity type, or in this case, a *Person*. Tables should be designed such that additions of information are easily made by adding rows, not columns, except for when some new type of attribute or variable about that entity type, needs to be added. In that case, defining a new column may be necessary. When creating a table, one specifies the name of table (usually reflecting the "entity type" it represents), and the names of the types of information contained in the columns. By definition, adding a new column to a table effectively creates a new table, whereas adding a row does not. This is because relational tables are defined by their name and the specific variables they contain.

Most current relational database management systems (RDBMS), including software such as PostgreSQL, MySQL, or Oracle—use SQL, or "Structured Query Language", to construct and manipulate complex tabular data. SQL is highly standardized such that the *Data Definition Language* (DDL) syntax for creating a Table will look very much like the following in many relational database management systems:

```
CREATE TABLE Seed_Traits
(Shape VARCHAR, Color VARCHAR, Count INTEGER);
```

The Table is given a name (*Seed_Traits*), and each Column (*Shape, Color, Count*) is given a name as well as a data type (*VARCHAR, VARCHAR, INTEGER*). The data type indicates how the computer should treat the cell values in those columns, e.g., as numerical ones amenable to arithmetic operations, or simply as text strings.

In "R" a similar table structure might be defined (as a data frame) as follows:

```
Seed_Traits <- data.frame(Shape=character(), Color=character(), Count=integer
());
```

Database (SQL)	Formal	Analytical term (R, SAS)
Table	Relation	Data frame, Dataset
Column	Attribute, Field	Variable
Row	Tuple	Record, Observation
Cell	Element	Value

Fig. 8.3 Terms describing table structures; read across rows for "synonyms"

Note the strong similarity in syntaxes between R and SQL—involving naming the table (called a data frame in "R") "Seed_Traits", and naming and typing the variables—Shape, Color, and Count. Despite these syntactical similarities, the underlying model of the data structure is somewhat different: a SQL table is a set of tuples/rows, while an R data frame is a list whose elements are vectors/columns. Technically, this leads to a difference in how data are inserted during the table/data frame creation process: in SQL—INSERT tuple/row; in R—add vector/column).

The DDL statements shown above for defining one's tables in SQL and R include the name of the table or data frame, names and types of the attributes, and how these, as well as tables, might be related to one another. These statements represent your *data model*, or *schema*, in executable code. It is best to use scripts such as these to define one's data since scripts can be later reviewed, modified, and re-executed, unlike, for example, simply typing your data into rows and columns in a spreadsheet. If one does not have the DDL statements (SQL, R, or other) for defining the structure and contents of one's tables, it is nevertheless important to have a good grasp of one's data schema, even if only represented in text or a diagram.

8.5.5 Tables or Spreadsheets?

Not everything that looks like a table *IS A* table. When one constructs a spreadsheet or worksheet in some popular application, such as Excel or Google Sheets, even if these are regularly structured in rows and columns with the first row being a "HEADER" row listing the column variables, one is typically not creating a true "table" by default. This is because there may be no "enforcement" of data typing—it would be permissible if one were to add a character string to a column in which all values should be numeric, or put a numeric value into a column that consists predominantly of text values. It is also common in many naively-constructed spreadsheets (and perniciously so) to find comments and other marginalia outside the "boundaries" of the table—which violate the integrity of the table and make it difficult to differentiate the tabular data from the other presentational, descriptive and summary elements surrounding it. While it is possible to use spreadsheets to create table-like structures, one should be aware that these other features can make later integration of those data highly problematic. One big advantage of using formal database software, like PostgreSQL or MySQL, or analytical software like "R" or SAS, is that when one creates a table or table-like structure (e.g., a data frame in "R"), the software will assist you in making the data conformant to the columns you specified with the data types you defined for them.

Exactly how tables are defined—with rows "collecting" data regarding one single object such as a person, or as often is the case in ecological and environmental studies, a heterogeneous entity, e.g., combining information about a specimen along with context gleaned from other sources, such as spatiotemporal data, information about the collector or methods, etc.—involves the art of *data modeling*. The choice of what variables to include in a table, or separate out into distinct

tables, depends to a great extent on the concerns of the researcher relative to the analyses they are planning for their data, and whether they might be "merging" their data with other data. However, it is always good practice to keep these principles in mind when constructing tables: to create tables that represent some entity type (e.g., a "person" table, or an "institution" table), structured with columns representing variables, and each row representing a set of "linked" or closely related (dependent) measurements. This will lead to data that are well constructed, and readily amenable to ingestion and further manipulation and analysis by most software packages. These are also the characteristics contributing to what is referred to in the "R" statistical world as "Tidy Data" (Wickham 2014).

8.5.6 Tables or Cross-classifications?

The cross-classification data format illustrated above in Fig. 8.2 superficially resembles a table. Where it differs is in how a well-modeled table can flexibly accommodate new values, as opposed to the cross-classification. In the cross-classification example above, the naming of "columns" as "Green" or "Yellow" creates a problem if a researcher wants to later track a new color, say "Light Green", or "Greenish-yellow". One can see that having a Variable for "Color" is better than using two possible *values* for Color (Green, Yellow) as *names* of Variables. With a well-constructed table, one simply adds Rows (records) to accommodate new values, e.g., for new SHAPES or COLORS (Fig. 8.4). Adding Columns, however, changes the table structure and, in many analytical packages, will require re-defining the table since you are adding a new Variable.

8.5.7 Modeling True Tables

Researchers often conceive of their data as residing in tables, but these tables often do not conform well with the formal requirement of representing some single entity type. Rather, researchers' tables are often inherently heterogeneous objects—coupling together (in a record or tuple; Fig. 8.3) information about several different

SHAPE	COLOR	COUNT
Smooth	Green	323
Wrinkled	Green	59
Smooth	Yellow	46
Wrinkled	Yellow	11
Smooth	*Light Green*	3
Semi-smooth	Green	1

Fig. 8.4 First few rows of Table "Seed_Traits" with more flexible structure than cross-classification depicted in Fig. 8.2

entity types or things—e.g., documenting the taxonomic identities, counts, and heights of **trees**; along with contextual information such as the **location** where those measurements were taken (e.g., place name, geo-coordinates); along with perhaps some **climatological data** (e.g., mean annual precipitation and maximum air temperature); and **additional metadata** (date of collection, data collector's name), etc.

Such *analytically-ready tables* are often products of prior data integration processes even if those integration processes were done manually—merging in ancillary data such as a place name or precipitation data that were in fact derived from other sources. Even these tables, however, should have ALL unique rows—if all the values are identical in two or more rows of a table, this would indicate either a duplication error, or that those seemingly identical records in fact represent measurements that should somehow be further differentiated. For example, if two records from the Table about "Seed_Traits" have identical values for the measurements of shape, color, and count, there should be some documentation, ideally contained in an additional column, about what "differentiates" those records—different sampling events in time, different specimen IDs, or measurements by different persons, etc.

While most scientists prefer working with *analytically ready tables* as above, *relational databases* take a different approach in modeling data tables, and for different ends. Relational databases are effective at storing complex and voluminous data, enabling a number of different analytically-ready tables to be derived from them. These analytically ready tables often result from the JOINing or integration of relational tables through a *query*. A *query* is a structured statement requesting information from the tables in a database and, as mentioned earlier, is often expressed in SQL. Relational databases are also very efficient at storing information with minimal redundancy. For example, information about taxonomic entities and their placement in a biological classification scheme (Family-Genus-Species) might be stored only once in a Table, but referenced many times by other tables in a relational database through a concise *Key relationship*, described in more detail below. Another advantage of a relational database is that, due to careful modeling and specification of relationships among tables through Keys and other constraints, the integrity of the data can be maintained, allowing multiple users to simultaneously add, delete, and update entries to the database without introducing errors (called "anomalies" in database terminology).

Data (as opposed to "statistical") normalization is another important concept that is perhaps not familiar to many researchers, unless they have had to model some fairly complex data. Nevertheless, it is a useful concept to know even when creating simple tables. Data normalization involves thinking about the entities that "naturally exist" in your data, and then separating these entities out into separate tables. Information that is closely associated with the same type of entity is kept together as a tuple in its own table—e.g., the information above about *Jane Smith* might be a record in a "People" Table. In database terms, these types of necessary and close associations among attributes are called "dependencies". When an attribute's value, e.g., an ORCID ID (ORCID, Inc. 2016), uniquely determines the value of some

other attribute (e.g., a birthdate), it is said that the birthdate is *functionally dependent* on the ORCID ID, or that the ORCID ID "functionally determines" the birthdate. Identifying functional dependencies in your data, and grouping these attributes together into the same table, is a big part of normalization. The most basic type of normalization, however, is called 1st normal form, and involves "atomizing" your data. Atomization is when you construct your tables such that the "cell" contents are limited to single values of "one thing" or measurement. Thus, it is not good practice to store values of an entire home address, or a given name and surname, or both latitude and longitude—together in a single cell. These should be separated out into more fundamental components—e.g., one variable (column) to hold values for latitude and one variable (column) to hold values for longitude.

"*Keys*" are extremely important in integrating data, as they are the set of values in a record that can be used to identify a unique occurrence or row in a table. Keys form the basis for linking one table with another by matching up their values *across* tables. Keys are still typically constructed such that they are only "unique" for rows within a specific database table. Local database keys are often generated as an additional column containing integer numbers, uniquely associated with a table row. A key can, however, also consist of more than one variable, if that is what is needed to uniquely identify a table row (e.g., both *date* and *location* might uniquely identify a record). Locally-scoped keys, however, make it difficult to identify and integrate compatible data *across* databases, which may be distributed around the Internet, or simply appear in multiple spreadsheets on a researcher's desktop computer.

There are two important roles that Keys play in data integration: as PRIMARY KEY, or FOREIGN KEY. For a given table, one can identify a PRIMARY KEY, which is the attribute or set of attributes that uniquely identify a record (row) *in that table* as distinct from all the others. Each row of a table must have a unique value for its PRIMARY KEY. For example, in a table called "People", that would contain one record per person, the ORCID Identifier could be specified as the Primary Key *for that table*. Wherever an ORCID Identifier might appear in *some other table*, however, it is called a "Foreign Key" and refers back to the table where that attribute is identified as its Primary Key. Thus, if an ORCID ID is designated as the Primary Key for a table containing a record for "*Jane Smith*", wherever that particular ORCID ID appears as a Foreign Key in another table, it will be clear that it refers to the one particular *Jane Smith*, and her associated attributes in the "People" table, and not records of other individuals with that same name of *Jane Smith*. In relational databases Key *relationships* among tables are formally specified by explicitly identifying which columns are Primary Keys or Foreign Keys, and called *constraints*. When using other analytical frameworks, such as "R" or SAS, however, Key constraints are often not explicitly specified, in which case the researcher must be fully aware of which variables can serve as Keys, in order to properly integrate data tables.

There is insufficient room here to describe in detail *relational algebra*, which provides the underlying theory for modeling relational data. There is a strong mathematical logic underlying how tables and their attributes are constructed

using those algebraic principles, how to model and normalize one's data, and how keys should be chosen and used to inter-relate tables for integration. While the syntactical and even structural aspects of creating, manipulating and integrating tables can vary across analytical frameworks, the theoretical bases of information modeling and relational database structures are still very broadly relevant, and can be found in many books that describe the theory and strategies for building databases (e.g., Halpin and Morgan 2008; Connolly and Begg 2014).

8.5.8 Need for Global Keys

As it becomes easier and easier to integrate data from multiple sources, there is a danger of re-counting certain observations—e.g., in the case where we might have two identical observations for a "Jane Smith", exhibiting the same height, weight, and birthday. These records might refer to two people with the same name, or be an erroneous repeat or "duplication" of data about the same "Jane Smith" individual, due to some earlier data integration. To differentiate whether these records refer to the same set of measurements or instances, one would need additional information, ideally involving unique identifiers, such as Social Security numbers (in the USA) or ORCID Identifiers.

Although the growth of the Web makes more data readily available to researchers for integration and analysis, it also makes more critical the need for "global keys"—in order to clearly reference the distinct instances of a multitude of objects, including not only obvious entities like people or institutions, but also other entities and phenomena that are singular—such as specimens from natural history museums or ice-core samples, or specific oceanographic cruises or field expeditions. In these cases, we might be able to infer from metadata what specific event or instance of some object took place or was measured—a survey of a vegetation plot, or a count of a herd of elephants—but in each case, we are relying on those metadata to differentiate one event or measurement from another. Critical information that allows such differentiation can often get lost when data are transferred, subsetted, summarized, re-combined, etc.

To afford better possibilities for data integration in general, and particularly over the Internet, Keys need to become *globally unique*. This requires expanding the notion of Keys beyond simply identifying unique rows in a database table, to also enable identifying unique tables, databases, and ideally any *distinct* information resource accessible over the Web. This is not as unfamiliar a notion as it first sounds—ORCID Identifiers essentially represent global Keys, in that they uniquely identify an instance of "something"—in this case individual researcher/scientists. This notion of a *globally unique identifier* may be conceptually simple, but its actual implementation can vary. The specific term "Globally Unique Identifier", or "GUID", for example, refers primarily to the assignment of a unique 128-bit digital signature to a digital object, with the capacity to assign over 10^{38} such unique addresses to items (probably enough to last us a while!). On the Web, however, a

Uniform Resource Identifier, or *URI*, can also serve as a globally unique identifier (Allemang and Hendler 2011). URIs have a similar format to the more familiar URL (*Uniform Resource Locator*) that we type into the address area of our Web browsers. These two differ in that URLs indicate a *location* of something on the Web, while URIs have a broader meaning indicating a *resource*, which is *any* object referenced and accessible through the Web, including simply "locations" (URLs) for Web sites or pages. Thus, all URLs are URIs, but not necessarily vice-versa. With the growing need to access and integrate distributed data across the Internet, URIs will increasingly serve as global Keys pointing to data resources across the Web.

Journal publishers are already issuing globally unique identifiers for articles, most typically as Digital Object Identifiers, or DOIs (International DOI Foundation 2016), while Social Security numbers and ORCID IDs are other examples of globally unique identifiers for individual persons. As we increasingly use URIs as globally unique identifiers, the URIs will also need to be persistent—in that there is an intention and commitment that these will always "point to" a specific resource that can be accessed over the Web. This process of accessing the value or other representation of the resource pointed to by a URI is called "dereferencing", and is a critical enabler of the Semantic Web and Linked Data (Berners-Lee et al. 2001; Heath and Bizer 2011). An increasing need for more effective integration of relevant data *distributed across the Internet* makes clear the advantage of using non-local, global Keys such as DOIs or ORCID IDs. Such global identifiers will become more commonplace in other aspects of scientific data and measurements, much more than simply persons or publications. Such new approaches for integrating Web-distributed data are not yet well covered in traditional references about databases, but are developing as blends of those technologies with emerging ones for the Semantic Web and Linked Data.

8.6 Merging or JOINing Tables

The most common motivation for data integration arises from researchers' needs to combine tables to create an enriched set of "coupled" variables that can be further manipulated and analyzed in search of various statistical relationships and other patterns. These techniques often require constructing records in which some of the variables are identified as "predictors", while other variables are hypothesized "outcomes" conditioned on the values of the predictor variables. Other motivations for integrating data might simply be to test for potential relationships among variables, such as positive or negative correlations with one another. These analyses rely on bringing the data together in ways that clarify which measurements are somehow associated together, which is often indicated by grouping those values together in the same record.

There are a number of ways to merge tables, depending on the needs and interests of the researcher. These various approaches have proper names and, even if a data

analyst often finds it necessary to mix and customize these approaches, it is good to be aware of the main integration processes and their proper names. To keep matters simple, the focus here will be on merging two tables together, but the same principles hold for merging together more than two tables. It is usually clearer, however, to merge tables in an iterative pair-wise fashion, at least initially.

The most basic way of merging or integrating two tables involves a Cartesian product, also known as a *cross join*, such that every row (or tuple) of one table is matched with every row of another table. That is, if Table A has 8 rows and 3 attributes, and Table B has 10 rows and 2 attributes, the resulting Cartesian product, Table C, has 80 (8×10) rows with 5 (3 + 2) attributes. However, it is rarely the case that researchers will be merging data in this way—usually doing so only if they need to create all possible combinations of the records from one data set with another. More typically, researchers are bringing data together based on some "matching" variable between the tables. These matching variables, which ideally are Keys, must be *domain compatible*—that is, they represent the same "thing" or measurement, and have the same set of allowable values. This generally requires an understanding of the semantic contents of the variables. If two Tables have records with the same value for a Key variable, this would indicate that those tables have some measurements in common.

8.6.1 APPENDING or Unioning

A common data integration operation used for combining two or more tables that have the same variables, is "appending". This involves simply attaching tables to one another, by matching along compatible variables, creating an output table that contains the sum of the rows from the input tables (minus potential duplicates). This is a highly useful operation when one collects numerous data sets of identical structure and column semantics, but that may be housed separately because, for instance, they are collected during different years, or from different places, or by different people. When appending these types of data, it is often necessary then, to afterwards add a column or two of additional information that will provide critical differentiating metadata about time, place, or person.

The two tables A and B in Fig. 8.5 both contain at least two variables of interest, for example, the names and weights of people participating on some project. In

Table "A"

ID	Name	wt
1	Amber	115
2	Bill	205
3	Dave	175

Table "B"

ID	name	wt
1	Mark	185
2	Matt	205
3	Rebecca	115

Fig. 8.5 Tables A and B, to demonstrate behavior of data integration operations

Table A our variables of interest are name and wt, while in Table B, the variables are also called name and wt. The ID variables in both tables A & B might be simple identifiers, that will not be of interest after the append unless additional information is added, since these will not be unique across tables after the tables are merged—they cannot serve as Keys for joining the two tables. To enable the ID column to retain its potential as a Key value *after* the append operation, we would need to modify the values, e.g., by including the originating table label as an identifier—hence, for Amber the identifier might be "A1", while for Rebecca it might be "B3". But, this would not happen automatically with a simple append operation.

In the standardized syntax of SQL, the following statement would be used to append these two tables:

```
CREATE TABLE C (ID, name, wt)
INSERT INTO 'C'
SELECT * FROM (
SELECT * FROM A
UNION ALL
SELECT * FROM B);
```

The "UNION ALL" would not exclude duplicate records from the result of combining the two Tables, whereas use of only a "UNION" statement would eliminate duplicate rows from the resulting table. The "SELECT *" statement here indicates that all the variables in the "FROM" table will be included in the output. Note that in this example we show the statements needed to create the output "Table C" using the SQL "CREATE TABLE", and "INSERT INTO" statements. In later examples we leave out this step, which in "R" would be like leaving out the assignment of an output to a new named data frame. The resulting output "Table C" appears in Fig. 8.6. Note that the "matching" structure of our two variables of interest allows us to simply vertically "stack" the two data sets to create one integrated output.

In "R", the *rbind* command very similarly appends data frames (assuming the Tables A and B above are converted to data frames A and B) if they have matching column names, and would produce the same output as above:

```
base::rbind(A,B) # using base R
dplyr::bind_rows(A,B) # using the dplyr package
```

In SAS, the *SET* command can be used for this and other powerful appending operations.

Fig. 8.6 Output Table C from appending the Tables A and B shown in Fig. 8.5

ID	name	wt
1	Amber	115
2	Bill	205
3	Dave	175
1	Mark	185
2	Matt	205
3	Rebecca	115

8.6.2 JOINs

Perhaps the most common data integration operations involve merging multiple tables based on matching values in specified subsets of variables shared among the tables. These types of operations are collectively called "JOINS" in SQL. Note, however, that other data manipulation frameworks, such as R, Matlab or SAS use different terminologies to refer to some of these data merging processes. For example, R has a number of named functions for doing these types of operations, and these vary from package to package (e.g., in R::base, or the popular *dplyr* and *data.table* packages). The R package *dplyr*, however, does use syntax largely borrowed from SQL for accomplishing many of its data frame "combining" operations. SAS accomplishes JOINs using both the *SET* and *MERGE* commands, and uses the sorted order of variables to determine the exact outcomes. In addition, however, SAS offers a rich set of SQL commands through its PROC SQL procedure.

We will describe four types of JOINs: INNER JOIN; and LEFT, RIGHT, and FULL OUTER JOINs. An understanding of how these joins work in SQL will provide a solid foundation for doing other types of merges, as is possible using powerful data manipulation languages such as R, SAS, or Python's *pandas* package.

When joining two or more tables, it is typically necessary to identify "matching" variables that represent the relationships or "linkages" among those tables. As discussed earlier, Key variables are often used for this purpose. Recall that modeling your data usually involves normalizing them—reducing data redundancy, clarifying the natural relationships among the data by grouping related attributes together into a Table, and identifying the Key variables. Ideally all functionally dependent information about an entity instance is fully documented in a single Table. For example, for a person named *Jane Smith*, all information tightly associated with that individual would be stored ONCE in a "People" Table —surname, given name, institutional affiliation, birthdate, etc. This greatly reduces the possibilities for errors because, e.g., if the values of any of these attributes need to be changed, the change only requires updating entries within one single record in one table. All other tables that might be linked to Jane Smith (and other peoples') record through a Key variable would then automatically be updated.

There are basically three ways in which records from separate tables can be related or "matched up", and these relationships are called *cardinality constraints*. Considering two tables at a time, records from one table to another can be related as: one-to-one (often indicated as "1:1"), one-to-many ("1:M"), or many-to-many ("M:N"). One-to-one relationships usually involve tables that are closely related, and often the attributes in these might appear in the same Table instead of stored separately. But sometimes there are reasons to separate these. For example, we might have a "People Public" Table with an ORCID ID as a *Primary Key*, while also containing other personal details (birthdate, country of birth) about some individual, but a separate "People Private" Table, that might contain a social security number, medical history or other confidential information. This latter table merely needs to contain a person's ORCID ID to be used as a *Foreign Key* to link back to ("match with") the appropriate Primary Key ORCID ID in the People

Public Table, or in this case, possibly vice-versa. Since every citizen and permanent resident of the USA should have one and only one Social Security number, and every scientific researcher should have an ORCID ID, there should be a 1:1 relationship among the records from those two tables.

More commonly, data tables are integrated through one-to-many relationships. For example, a relationship linking ("matching") taxonomic names (stored in a Taxonomy Table) to the taxonomic identity of individual plant stems recorded in a Vegetation Plot Table is 1:M, since a single taxonomic name can be assigned to many stems, but any given stem can only be assigned to a single taxon. One can reverse the ordering of this description—many stems can belong to one taxon, and describe that relationship equivalently as M:1. In contrast, the relationship between a Table listing plant host species and a Table of potential pollinator species would likely be many-to-many (M:N), since a plant might host multiple pollinator species, and a pollinator might visit multiple plant species. Note finally that there may also be no relationship at all between the records in two tables: a table containing measurements of plant heights from sampling plots in Kansas might have no relationship with a table of measurements of barnacle densities around deep-sea hydrothermal vents in the Pacific Ocean! These two tables might have no *Key variables* in common that could provide a basis for matching up and integrating records across them.

Of these potential relationships, integrating tables that have many-to-many relationships with one another are the most complex, and require careful planning in order to create effective and meaningful joins. In relational database systems where the data are highly normalized (i.e., data are atomized, redundancies minimized, and functionally dependent attributes are in the same tables), these situations often require creation of another table, variably called a "linking", "associative", "junction", or "mapping" table, that essentially includes the Key mappings of the matching instances from both of the tables. For example, a familiar situation is the many-to-many relationship between books and their readers (books are read by multiple people; people read multiple books). The *linking table* would contain individuals' ORCID IDs in a tuple with the ISBNs (a type of global identifier) of the books they've read. There would be a record for each "reader/book" combination. Thus the table would contain multiple records per person, as well as multiple records per book. But the person and book tables, if normalized, would each contain only one record per person or book, respectively. The linking or associative table is necessary to document the specific many-to-many relationships among entries from the two normalized tables. Creation and use of such tables in relational databases is a topic that is well covered in books on modeling relational data (Halpin and Morgan 2008; Connolly and Begg 2014).

It is common in many field-collected ecological data sets, however, for the raw data to be entered "as observed", such that the same taxonomic names might appear in many records, such as in the plant/pollinator case described above. These are essentially already "linking tables" that allow one to bring together, as 1:M JOINs, data from normalized tables that might describe, e.g., the traits of the plant species (Plant Trait Table; 1 row per taxon), and the traits of the pollinator species (Pollinator Trait Table; 1 row per taxon). One should carefully consult the manual for one's software to be sure of the exact syntax needed for merging tables that have many-to-many matches.

Here we focus on the simpler but very common cases that involve merging tables with 1:1 and 1:M relationships. The latter case is particularly useful when integrating tables based on key relationships that bring together complementary data that enable exploration of new hypotheses and analyses.

The first type of JOIN we will describe is called an INNER JOIN, in that it includes only those rows that are matches of variables from both tables. Here, we do an INNER JOIN on the variable prodID (which might represent a purchase code in this example), found in Tables Customer and Item (Fig. 8.7) using SQL code:

```
SELECT cName, prodID, pName, cost
FROM Customer AS C
INNER JOIN Item AS I
ON C.prodID= I.prodID
```

resulting in the output shown in Fig. 8.8.

- The "**Customer AS C**" statement enables the name of the table to be abbreviated from "Customer" to later be referenced as simply "C"
- In the statement "**ON C.prodID**", the prodID attribute in table Customer is being referenced.
- The "**ON**" statement references the same "linkage" variable, prodID, from both the Customer and Item tables.

Note that the record where C.prodID=2500, ID=3 and Name="Dave" from Table Customer is missing from this output; as is the row where P.prodID=25, and Name="Cheese" from Table Item. That is because there were no matches for the values of the prodID variable of those records (C.prodID=250 or P.prodID=25) in both tables. In this case, Customer Dave may not have bought Item Cheese, but instead purchased something else that was not listed in the Item table.

Table "Customer"

ID	cName	prodID
1	Amber	15
2	Bill	30
3	Dave	2500
1	Amber	201
2	Bill	215

Table "Item"

prodID	pName	cost
15	Crackers	24
25	Cheese	48
30	Wine	89
201	Plates	10
215	Cups	12

Fig. 8.7 Sample Tables "Customer" and "Item" to demonstrate behavior of different data integration operations

cName	C.prodID	I.prodID	pName	cost
Amber	15	15	Crackers	24
Bill	30	30	Wine	89
Amber	201	201	Plates	10
Bill	215	215	Cups	12

Fig. 8.8 Output from INNER JOIN on Tables "Customer" and "Item" from Fig. 8.7

You could accomplish this type of inner join in "R" using the *merge* command on analogous data frames of Customer and Item:

base::merge(Customer, Item, by.x="prodID", by.y="prodID", all=FALSE)
dplyr::inner_join(Customer, Item, by="prodID")

For ecologists, one might imagine the above tables linking taxonomic names of Predators with their Prey Items through a matching "food_for_ID" (or "eats_ID") variable. Or a table consisting of "Stream Chemistry" measurements might be linked through a variable such as "locationID" to a "Site" table that included details about the place that was sampled (stream name, geo-coordinates, mean flow volume, stream depth and width, etc.). In order not to distract researchers about the "correctness" of the measurements and entities JOINed in these examples, we keep them very generic. We challenge the scientist to imagine how these JOIN patterns pertain to their own data integration needs.

Another common case is when someone wants to match a variable or variables in one table with those in another, but doesn't want to "lose" those records that have no match. Instead, the unmatched values might have missing or NULL values for any additional variables represented in the newly merged output table. These are OUTER JOINs, and there are three types: LEFT, RIGHT, and FULL.

A LEFT (OUTER) JOIN, often called a "LEFT JOIN" follows—

SELECT cName, prodID, pName, cost
FROM **Customer AS C**
LEFT JOIN **Item AS P**
ON C.prodID= P.prodID

resulting in the output shown in Fig. 8.9.

The values of "NULL" for variables pName and cost in the resulting output Table occur where the attributes had no matches for C.prodID=P.prodID for value=2500 in Tables Customer and Item. The "LEFT JOIN", however, specifies that even unmatched records from the first (LEFT-hand) mentioned table (here Table Customer with cName="Dave" and C.prodID=2500), are carried into the resulting output table, so every row from Table Customer appears in the output Table.

Similar results using "R" could be generated as:

base::merge(Customer, Item, by.x="prodID", by.y="prodID", all.x=TRUE)

or

dplyr::left_join(Customer, Item, by="prodID")

Fig. 8.9 Output from LEFT JOIN on Tables "Customer" and "Item" from Fig. 8.7

cName	C.prodID	P.prodID	pName	cost
Amber	15	15	Crackers	24
Bill	30	30	Wine	89
Dave	2500	NULL	NULL	NULL
Amber	201	201	Plates	10
Bill	215	215	Cups	12

Note that the exact special characters or numbers representing a "NULL" value in a table (or data frame) can vary depending on the database or analytical framework you are using. Also note again that the ordering of rows in a table is arbitrary—a table can be sorted according to the values for any variable of interest, but by definition there is no intrinsic ordering of the rows (or columns for that matter) in a table.

A "RIGHT JOIN" is very similar to a LEFT JOIN, only differing in that ALL the records from the second mentioned (RIGHT-hand) Table are retained, with unmatched variables from the first (LEFT) table filled with NULL values:

```
SELECT cName, prodID, pName, cost
FROM Customer AS C
RIGHT JOIN Item AS P
ON C.prodID= P.prodID
```

resulting in the output shown in Fig. 8.10.

In the case of OUTER JOINs, the exact nature of the output can also vary, depending on what analytical package you are using and the options you specify for the results—such as whether both the "prodID" variables from the "left" and "right" tables are included in the result set or not.

One would produce very similar results in "R" using:

```
base::merge(Customer, Item, by.x="prodID", by.y="prodID", all.y=TRUE) # BASE
dplyr::right_join(Customer, Item, by="prodID")
```

A "FULL OUTER JOIN" maintains ALL the records from both Tables, and fills in unmatched variables with NULL values as shown in Fig. 8.11.

```
SELECT cName, prodID, pName, cost
```

Fig. 8.10 Output from RIGHT JOIN on Tables "Customer" and "Item" from Fig. 8.7

cName	C.prodID	P.prodID	pName	cost
Amber	15	15	Crackers	24
Bill	30	30	Wine	89
Bill	215	215	Cups	12
NULL	NULL	25	Cheese	48
Amber	201	201	Plates	10

Fig. 8.11 Output from FULL JOIN on Tables "Customer" and "Item" from Fig. 8.7

cName	C.prodID	P.prodID	pName	cost
Amber	15	15	Crackers	24
Bill	306	306	Wine	89
NULL	NULL	25	Cheese	48
Amber	201	201	Plates	10
Dave	175	NULL	NULL	NULL
Bill	215	215	Cups	12

FROM **Customer AS C**
FULL JOIN **Item AS P**
ON C.prodID= P.prodID

In "R", near equivalent result sets can be created by:

base::merge(Customer, Item, by.x="prodID", by.y="prodID", all=TRUE)
dplyr::full_join(Customer, Item, by="prodID")

The attentive reader will note that the above JOINs also demonstrate one-to-many relationships of the Customer with the Item table—Customers Amber and Bill both are associated with multiple Items. Also, one can usually imagine some implicit "verb" that describes the relationship between tables that are JOINed. For example, in this case, the relationship might be "purchases". Amber purchases Crackers and Plates, while Bill purchases Wine and Cups.

Be aware that different analytical packages can handle JOINs in very different ways, both in terms of how to specify the desired operation in some computing language (the syntax), as well as variation in how the outputs are presented. For example, the R *base::merge* function operates very differently, both syntactically and semantically, from functions in the *dplyr* or *data.table* packages, for doing join operations. However, as can be seen here, even R's *base::merge* function is quite flexible and powerful for doing various JOINs. It is necessary to read the manual for your software very carefully when doing JOINs, to be sure that your outputs are what you expect them to be.

8.7 The Datum Is the Atom

While tables represent one of the most common and useful structures for storing and integrating data, it is important to understand at the most atomic level: what are the phenomena being named, typed, described, and/or quantified and preserved in digital data? The growing demand for synthesis of data across traditional scientific disciplines further motivates the need to better understand what data and observations are collectively available, so these can be integrated for further analysis. These trends have led many earth and biological science informatics groups to recently converge on a common model for data: as sets or collections of *observations and measurements* (Madin et al. 2008; Cox 2015). These models all seek to enable more efficient data integration by placing the focus on data at its most elemental levels, which are the individual observations and measurements. Scientific observations are decomposed into constituents representing the ***entity*** (thing or process) that is observed; the ***characteristics*** of the entity or process that were documented or measured, and assigned values; and the specific scale or ***units*** associated with those values (Fig. 8.12). In many cases, additional critical metadata

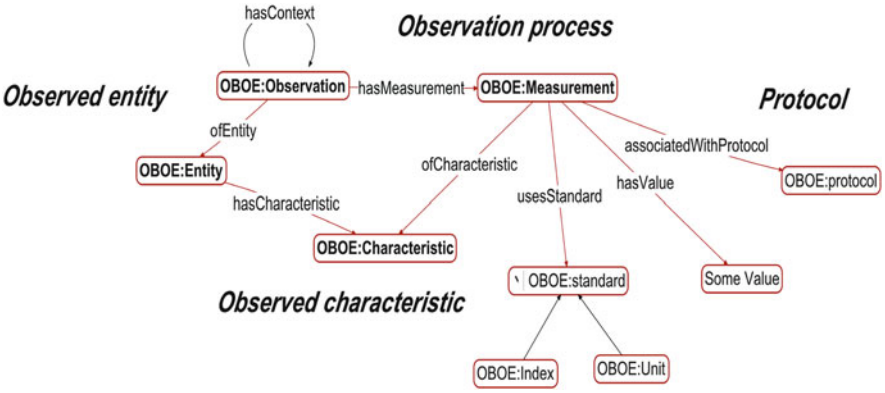

Fig. 8.12 Schema for a typical observational data model

might describe methodological or protocol specifications as well (although these often might not be available, or readily inferred).

While the most common use of the term "observation" still refers to a row or record in a table (e.g., SAS uses this terminology), in many cases calling these row-level accumulations of data "observations" is somewhat misleading. As we have seen, individual values "coupled together" in a record often come from multiple original data sources, and are often associated together by some researcher for some specific purpose, such as for a regression analysis. In most observational data models, the observation is highly atomic—representing an act of measurement of some particular instance, individual, or phenomenon. Then, as depicted in Fig. 8.12, these observations are inter-related through some "has Context" relationship, where "has Context" can be further clarified by using a more specific description, such as "has same geospatial context" or "taken from same specimen".

For example, a table record might consist of seven observations: (1) mean daily air temperature coupled with (2) monthly precipitation measurements (acquired from a nearby set of weather gages), merged with weekly censuses of the (3) names and (4) abundances of herbivorous insect larvae found on some (5) named plant species, located at (6) some geospatial place name or geo-coordinates, at (7) some point in time. In this case, our table really consists of only five distinct "entities", some with many potential characteristics that may or may not have been measured or otherwise documented, e.g., (1) characteristics of the local atmospheric conditions, (2) traits of the insect larvae, (3) traits of the plant host species, (4) descriptive aspects of the local habitat and associated terrestrial environment, and (5) the date-time. We could call such a "composite" record a single observation (according to some relational data terminologies), **or** using the terminology of most *observational data models*—it would consist of several linked or coupled observations or measurements—consisting of the *physical characteristics* of some place, that contextualize some measurements on *insect* and *plant entities*, with further information about a place's *geospatial location* (Fig. 8.13). We have also depicted in Fig. 8.13 how specific measurements might be annotated via a unique identifier, in

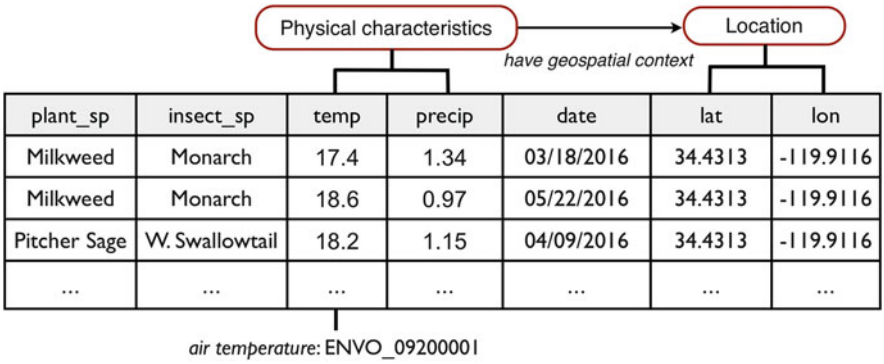

Fig. 8.13 Observational data model showing relationships among table variables, and reference to external identifier

this case clarifying that the column labelled "temp" measures an *air temperature*, as defined by the Environment Ontology's term ENVO_09200001, which references a URI (University of Michigan 2016a). The base URI for the "air temperature" measurement in the Environment Ontology would be "http://purl.obolibrary.org/obo/ENVO_09200001", where the "purl" in the address indicates that this is intended to be a *persistent* URL or "PURL", that is also globally unique.

Careful examination of the "entities" in Fig. 8.13 reveals that for each, only a small set of potential "characteristics" were measured—for weather, only air temperature and precipitation (Fig. 8.13), and not, e.g., relative humidity or air pressure, but these might be linked in from other sources. It further becomes clear that there must be information as to the *units* associated with many of these measurements, e.g., degrees *Celsius* for air temperature, or *centimeters* for precipitation. All recorded values in scientific data should use rigorously defined, highly comparable standards for measurement, as much as possible, to enable consistent, accurate interpretation by researchers regardless of language, location, or time of access.

Regarding the plant/herbivore data in this example table—additional attributes might result from direct measurement such as plant size or age, but others might be inferable simply from knowledge about an entity's belonging to some class. For example, one of the plant species might be identified as a Milkweed, and through that name linked to an entry in another table providing more complete taxonomic information, such as the scientific name *Asclepias spp*., as well as plant family, tribe, etc. Milkweed is poisonous to grazing animals, but the toxicity and potential avoidance of Milkweed plants by specific insect species might not be known to the researcher, nor directly assessed by them. Curiosity about how plant chemistry might be structuring plant/insect interactions could motivate further integrating these data with records from another table (possibly prepared by another researcher)—that documents the toxicity of specific chemicals to various insect taxa. From this integrated output table, we might discern that monarch butterflies and several other insect species are immune to Milkweed toxins, but a number of

insect species are not. One can see that many data integration actions are motivated by importing (through a JOIN) a subset of columns from other data sets. The observational data model makes clear that each value in a column represents a distinct observation or measurement, and that the full tuple (row or record) structure in a table might result from integration of multiple heterogeneous data sources, coupled together to help inform some particular analysis of interest.

Observational data models afford maximum flexibility with regards to defining tables by encouraging explicit specification of the semantics necessary to understand the contents of any table "cell" value—in terms of the entity of interest, the characteristics of the entity that are measured, and the units and methods involved in obtaining the measurement(s). This type of information might be contained in a *Data Dictionary*, or more recently, documented using various *metadata standards*, such as the Ecological Metadata Language, EML (Fegraus et al. 2005), that is used by environmental data repositories such as the U.S. National Science Foundation's Arctic Data Center (2016), or DataONE (2016). Ironically, however, there are numerous ways in which the "same" entities and their characteristics can be described or defined in data dictionaries or metadata standards, making evaluation of their semantic similarity or equivalence difficult. In the same way that standardization of measurement units, such as agreement on the meaning of a "millimeter" or "kilogram", has greatly facilitated comparability of data, there is currently a need to increase standardization of the names of entities and measurements, so that scientific data are more effectively discoverable, interpretable, and amenable to potential integration with other data.

One potential solution to reducing the current "babel" of scientific terminologies involves developing and agreeing upon community vocabularies. Entire data objects, as well as individual measurements, are increasingly semantically defined in this way, by referencing terms in web-accessible vocabularies (known as "ontologies") through dereferenceable globally unique URIs. For example, if one wants to indicate that the measurements in the table depicted in Fig. 8.13 were taken from a "Mediterranean grassland biome", one can indicate that by referencing a persistent URI, such as one found in the Environment Ontology, ENVO (Buttigieg et al. 2016; University of Michigan 2016b). Syntaxes used to define data in this way are typically expressed in Resource Description Framework (RDF) and Web Ontology Language (OWL)—which are the formal languages recommended for the Semantic Web (Berners-Lee et al. 2001; W3C OWL Working Group 2016). The exact mechanisms for linking ontology terms to digital object structures, e.g., tables, table columns, or even cell values in databases, are currently under development, and involve a process known as "semantic annotation" (Madin et al. 2008). In any case, when it becomes easier and more common for researchers to reference terms from shared vocabularies to unambiguously describe their data via Web-based globally unique identifiers such as URIs, data integration will become much easier. In addition, this practice will enable more precise semantic searches, such that measurements from multiple tables of data distributed around the Web can be more effectively discovered, and "matched" with other measurements.

8.8 Conclusion

Data integration involves bringing together distinct, often heterogeneous data, in order to enable more powerful and comprehensive modeling and analysis of phenomena of interest. Data integration is a key data manipulation process that is driving much scientific synthesis by enabling the coupling together of additional and complementary information to derive more holistic or robust understanding.

Our focus here has been on tabular data—one of the most common data structures that field scientists in particular must contend with. We have tried to convey that there is a strong theoretical basis for constructing and interpreting data (for tables in particular). We have emphasized here some of the basic principles, and encourage the reader to investigate the subject further, as this will not only enhance understanding of tabular data structures, but should lead to more robust and accurate coding of diverse data integration processes, that can often get quite complex.

The specific syntactical mechanisms for integrating data can vary considerably from one analytical package to another, but the relational data model and the standardized SQL syntax provide a strong foundational perspective for understanding how to flexibly create, transform and merge tabular data. This understanding can transfer into better practice as well when using free-form tools such as spreadsheets to capture and manipulate scientific data. We hope, however, the reader is convinced that more specialized data creation and manipulation tools that require explicit specification (e.g., in coded statements) of table structures, their key attributes, and how tables are related to one another provide huge advantages with regards to the flexibility and transparency of creating and operating on one's own, as well as others' data, in both the short and long term.

Finally, it is important to recognize that scientific data are collections of "assertions" about the state of physical reality at some time and place, taken by observers who are trained in appropriate, objective methods, often using sophisticated, specialized instruments. As such, scientific data should be regarded with special respect, and in many cases, carefully preserved in ways that can be accessed and interpreted in the future. This is particularly true for data about the state of the environment, which we now know to be rapidly changing. Well-conceived, well-structured records about the state of our environment in the past, present, and into the future—of biological, physical, chemical and other measurements—may prove invaluable for understanding both the long-term and short-term processes that are governing the state of our biosphere. Facilitating the discovery and integration of these records through sound data management practices will provide great benefits to the scientific research enterprise when trying to uncover the complex relationships that govern ecosystems and their components in this age of increasing global human impacts.

Acknowledgements I would like to thank Julien Brun for suggesting several useful changes and corrections to the text. Shawn Bowers was a stalwart companion while dissecting the structure of numerous scientific datasets. But I want to acknowledge especially many years of fruitful and stimulating discussions with Matthew B. Jones on matters regarding the nature of ecological data,

and the need for better software tools and cyberinfrastructure to support synthesis and collaboration in the environmental sciences. The National Center for Ecological Analysis and Synthesis, NCEAS, has provided a strongly supportive environment for advancing ecoinformatics practice, and still represents, to my mind, a beacon for promoting and facilitating synthesis in the ecological and conservation sciences. Finally, I want to thank colleagues from several past and ongoing NSF-sponsored Cyberinfrastructure projects, including DataONE (NSF #1430508), SEEK (NSF #0225676), SONet (NSF #0753144), and the KNB (NSF #9980154). It has been a continual and pleasurable collaborative learning process with many bright and selfless colleagues.

References

Allemang D, Hendler J (2011) Semantic web for the working ontologist, Effective modeling in RDFS and OWL, 2nd edn. Morgan Kaufmann, Waltham, MA

Arctic Data Center (2016) NSF Arctic Data Center. https://arcticdata.io. Accessed 5 Dec 2016

Berners-Lee T, Hendler J, Lassila O (2001) The semantic web: a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. Sci Am 284:34–43

Buttigieg PL, Pafilis E, Lewis S et al (2016) The environment ontology in 2016: bridging domains with increased scope, semantic density, and interoperation. J Biomed Semantics 7:57. doi:10.1186/s13326-016-0097-6

Canfield MR (ed) (2011) Field notes on science and nature. Harvard University Press, Cambridge, MA

Carpenter SR, Armbrust EV, Arzberger PW et al (2009) Accelerate synthesis in ecology and environmental sciences. BioSci 59:699–701. doi:10.1525/bio.2009.59.8.11

Codd EF (1970) A relational model of data for large shared data banks. Commun ACM Classics 13:377–387. doi:10.1145/362384.362685

Codd EF (2000) The relational model for database management: version 2. Addison Wesley, Reading, MA

Connolly T, Begg C (2014) Database systems: a practical approach to design, implementation, and management, 6th edn. Pearson, Upper Saddle River, NJ

Cook RB, Wei Y, Hook LA et al (2017) Preserve: protecting data for long-term use, Chapter 6. In: Recknagel F, Michener W (eds) Ecological informatics. Data management and knowledge discovery. Springer, Heidelberg

Cox S (2015) Ontology for observations and sampling features, with alignments to existing models. http://www.semantic-web-journal.net/content/ontology-observations-and-sampling-features-alignments-existing-models-0. Accessed 5 Dec 2016

Darwin C (1837) Darwin's first diagram of an evolutionary tree *from* First Notebook on "Transmutation of Species". https://commons.wikimedia.org/wiki/File:Darwins_first_tree.jpg. In the public domain {{PD-US}}. Accessed 24 Jan 2017

DataONE (2016) DataONE. https://www.dataone.org. Accessed 5 Dec 2016

Fegraus E, Andelman S, Jones MB et al (2005) Maximizing the value of ecological data with structured metadata: an introduction to Ecological Metadata Language (EML) and principles for metadata creation. Bull Ecol Soc Am 86:158–168. doi:10.1890/0012-9623(2005)86[158:MTVOED]2.0.CO;2

Greene B (2005) The fabric of the cosmos. Vintage, New York

Halpin T, Morgan T (2008) Information modeling and relational databases: from conceptual analysis to logical design, 2nd edn. Morgan Kaufman, Burlington, MA

Hampton SE, Anderson SS, Bagby SC et al (2015) The Tao of open science for ecology. Ecosphere 6:1–13

Heath T, Bizer C (2011) Linked data: evolving the web into a global data space (1st edn). Synthesis lectures on the semantic web: theory and technology 1:1, 1–136. Morgan & Claypool, London

Hempel C (1970) Aspects of scientific explanation, and other essays in the philosophy of science. The Free Press, New York

International DOI Foundation (2016) https://www.doi.org. Accessed 5 Dec 2016

Kuhn T (1996) Structure of scientific revolutions. University of Chicago Press, Chicago

Madin J, Bowers S, Schildhauer M et al (2008) Advancing ecological research with ontologies. TREE 23(3):159–168. doi:10.1016/j.tree.2007.11.007

Michener WK (2015) Ten simple rules for creating a good data management plan. PLoS Comput Biol 11(10):e1004525. doi:10.1371/journal.pcbi.1004525

Norvig P (2009) Natural language corpus data, Chapter 14. In: Segaran T, Hammerbacher J (eds) Beautiful data: the stories behind elegant data solutions. O'Reilly Media, Sebastopol, CA, pp 219–242

ORCID, Inc. (2016) ORCiD. http://orcid.org. Accessed 5 Dec 2016

Platt JR (1964) Strong inference. Science 146(3642):347–353. doi:10.1126/science.146.3642.347

Quine WV (1981) Theories and things. Belknap Press of Harvard University Press, Cambridge, MA

Strasser C, Abrams S, Cruse P (2014) DMPTool2: expanding functionality for better data management planning. Int J Digital Curation 9(1):324–330. doi:10.2218/ijdc.v9i1.319

Taper ML, Lele SR (eds) (2004) The nature of scientific evidence: statistical, philosophical, and empirical considerations. The University of Chicago Press, Chicago

University of Michigan (2016a) Ontobee: environment ontology. http://www.ontobee.org/ontology/ENVO?iri=http://purl.obolibrary.org/obo/ENVO_09200001. Accessed 5 Dec 2016

University of Michigan (2016b) Ontobee: environment ontology. http://www.ontobee.org/ontology/ENVO?iri=http://purl.obolibrary.org/obo/ENVO_01000224. Accessed 5 Dec 2016

W3C OWL Working Group (2016) Web Ontology Language (OWL). https://www.w3.org/2001/sw/wiki/OWL. Accessed 5 Dec 2016

Wickham H (2014) Tidy data. J Stat Softw 59(10):1–23. doi:10.18637/jss.v059.i10