[DM-01-035] Logical Data Models

Abstract

A logical data model is created for the second of three levels of abstraction, conceptual, logical, and physical. A logical data model expresses the meaning context of a conceptual data model, and adds to that detail about data (base) structures, e.g. using topologically-organized records, relational tables, object-oriented classes, or extensible markup language (XML) construct tags. However, the logical data model formed is independent of a particular database management software product. Nonetheless such a model is often constrained by a class of software language techniques for representation, making implementation with a physical data model easier. Complex entity types of the conceptual data model must be translated into sub-type/super-type hierarchies to clarify data contexts for the entity type, while avoiding duplication of concepts and data. Entities and records should have internal identifiers. Relationships can be used to express the involvement of entity types with activities or associations. A logical schema is formed from the above data organization. A schema diagram depicts the entity, attribute and relationship detail for each application. The resulting logical data models can be synthesized using schema integration to support multi-user database environments, e.g., data warehouses for strategic applications and/or federated databases for tactical/operational business applications.

Keywords: database design

Author & citation

Nyerges, T. (2017). Logical Data Models. The Geographic Information Science & Technology Body of Knowledge (1st Quarter 2017 Edition), John P. Wilson (ed.).
DOI: 10.22224/gistbok/2017.1.2

This Topic is also available in the following editions: DiBiase, D., DeMers, M., Johnson, A., Kemp, K., Luck, A. T., Plewe, B., and Wentz, E. (2006). Logical models. The Geographic Information Science & Technology Body of Knowledge. Washington, DC: Association of American Geographers.

Explanation

1. Definitions

data model: There are many definitions of a data model, but there are two main perspectives. The most comprehensive definition of a data model comes from Edgar Codd

(1980): A data model is composed of three components: 1) data structures, 2) operations on data structures, and 3) integrity constraints for operations and structures. A second perspective arises from James Martin (1976) and others who developed the idea of data(base) structure diagrams. As such, it aligns with the first component articulated by Codd (1980), and in this simplicity, is the more popularized version. We use the more popularized version herein. However, the second and third components of Codd (1980) are necessary for full understanding of how data and data derivations become information.

logical data model: A translation of the conceptual data model content into structures implementable by database management software, but not dependent on a particular database software (Simsion & Witt, 2005, p. 17)

2. Data Models, Data Modeling, and Logical Data Modeling

From a data management perspective, a data model defines the structure and intended meaning of data (West, 2011, p. 5). A more detailed view characterizes a data model as consisting of (Codd, 1980, p. 112): 1) a collection of data structure types (the building blocks of any database that conforms to the model); 2) a collection of operators or inferencing rules, which can be applied to any valid instances of the data types listed in (1), to retrieve or derive data from any parts of those structures in any combinations desired; 3) a collection of general integrity rules, which implicitly or explicitly define the set of consistent database states or changes of state or both -- these rules may sometimes be expressed as insert-update-delete rules. The perspective offered by West (2011) is essentially the first component offered by Codd (1980). In this description of logical data models we focus on the view given by West (2011), but point out that Codd's (1980) perspective provides useful insight for understanding software design for data management systems, encompassing the database design perspective given by West (2011).

Data modeling describes a workflow process that translates information requirements into a fully implementable database design. Three levels of data model abstraction – conceptual, logical, and physical -- represent products of a database design process (Simsion & Witt, 2005). The focus herein is on a logical data model. In the process of database design, a logical data model adopts the data meaning context of the conceptual model, while introducing logical data structuring. Once a logical data model is designed, it can be translated into a physical database design implementation using a particular database management system (DBMS), which organizes the implementation of a database.

Logical data structuring often follows language formalisms, e.g., formalization using set theory or graph theory in mathematics. Language formalisms enhance expressiveness and robustness beyond data meaning, although the language components could follow the convenience of expression from any language. Whichever approach is used, data model components involve data language constructs, e.g., topologically-organized records, relational tables, object-oriented classes, or XML construct tags. As such, a variety of types of logical data models (data languages) exist for organizing geospatial data constructs. Among the more popular types of DBMS logical data models for geospatial data are the following.

- Relational – set theory with relational calculus and implemented using relational algebra; relation tables are formed as collections of fields

- Relational Column Store – founded on relational data model with extensions focused on column-wise fast data retrieval
- Object-oriented – data structured using phenomena-centric objects with customized operations relevant to phenomena
- Graph – graph theoretic foundation for characterizing nodes and edges
- NoSQL – no structured query language, aka non-schema language, key-value store focuses on variable number of field types per record; document store is a sub-class focusing on documents; both of these can use resource description framework specification based on ontology
- Open Standards – Open Geospatial Consortium simple features, and other features organized as data services (collections of instances) rather than individual data instances
- XML – XML tag specifications developed for a knowledge domain, perhaps using a substantive, e.g. transportation, domain for specifying the character of features in relation to one another

All of the above approaches organize, read, interpret and manipulate data constructs and data using a data language. A data language is composed of at least two sub-languages for description and manipulation. A data description language manages (create, read, update, delete) constructs. The expression of constructs is called a schema. A data manipulation language manages (create, read, update, delete) database instances organized in terms of the schema constructs. The data languages are formulated independent of programming language constraints; but nonetheless constrained by the class of constructs and operations within the language formalism. Some software designers claim that noSQL (non-schema) data management systems foster unstructured data organization and storage, or at least foster an approach that distributes the description to data. Nonetheless, herein we consider the nature of a logical data (schema) description language, leaving the data manipulation language for an entry about data management software.

3. Components of a Logical Data Model Schema

Given the availability of a variety of data language formalisms as indicated above, and despite the construct constraints associated with each type of formalism, it is possible to identify a limited set of construct types that generalize across logical data models, while drawing from the conceptual data model level. Entity types, attributes and relationships are the most common constructs among different types of logical data models as described below. However, it is important to remember that formalisms of a particular data model type can provide additional detail to logical structuring, and could be used to nuance components for a logical data model schema.

3.1 Structuring Entity Types with Attributes Organized for Logical Completeness

The main focus of logical data organization involves representing the details of entity types with attributes. A major challenge in logical modeling is to offer complete representation of data classes, while at the same time avoid redundant representation through duplicative reference where it is not intended. In contrast, a complex entity type is one that hides other entity types, and should be unpacked into the separate types using type and sub-type referencing. Type hierarchies organize different levels of meaning for entity classes and the

corresponding entity types that define those classes. For example, a property parcel can be seen as two types, a tax parcel (the purpose of which is levying taxes) and a development parcel (the purpose of which is to assign permits for changing the character of the parcel). Both types retain the primary attribute of owner, but taxation of owner, and providing an owner a right to develop the property are different activities that need be separated because they are managed (commonly) by different units of a jurisdiction (.e.g., township, city, county etc.). The types are established through attributes, commonly primary attributes, of the entity type.

3.2 Primary and Secondary Attributes for Entity Classes

Entities as instances need be uniquely accessible through a primary attribute of the class (see conceptual data model). Entities represented by records should have an internal (primary key) identifier within a database and could/should be managed as unique. Relationships should not be used as part of the internal identifier. Secondary attributes provide further detail for characterization of entity classes. These are descriptors that are useful but not necessary to the identification of entities in the class. These descriptors can also have keys associated with them, but they would not be primary leys. Whether primary or secondary, every attribute is measured on some dimension. That dimension is a quality for which the attribute characteristic is measurable using some unit of measurement. The dimension can include the complete collection of data values that could be specified, and these data values are often differentiated in terms of a level (scale) of measurement, e.g. nominal, ordinal, and interval ratio (Stevens 1946). A nominal unit of measurement is a categorical identifier, e.g. name. An ordinal unit of measurement such as integer rank can be used to provide indication of more or less of some amount. An interval measurement provides magnitude across a range of numbers. A ratio measurement provides a normalized characterization across a range between 0 and 1. Chrisman (2002) extended Stevens four levels into eight levels to show that GIS measurements are sometimes more nuanced than previously thought, but the four are still fundamental. A domain for a dimension need be specified, and these domains are guides for how data will be represented in software. A domain is the collection of data values that will be found within the fields of the attributes. A dimension can be restricted to a range of data values, which is a restricted domain.

Each of the entity classes within a logical model can be stored in one or more tables depending on the need for table normalization that involves attributes of an entity class. Normalization is a logical structuring technique, and can influence the integrity of data access. A table (relation) is in first normal form if and only if the domain of each attribute contains only primitive (indivisible) values, and each attribute value containing a single value only from that domain. A table is in second normal form if it is in first normal form and no non-prime attribute (not a part of any candidate key) is dependent on any proper subset of any candidate key of the table. A table is in third normal form if it is in 2nd normal form and when the table is free of insertion, update, and deletion anomalies due to access uncertainty to those data elements.

Spatial data types describing geometric shapes, and coordinate specifications therein, are described at the logical level. Points, lines, polygons, and raster cell types are all part of the spatial data types.

3.3 Relationships

Details for logical structuring characterize how relationships among data units are to be organized. Relationships should be used to express the involvement of entity types with activities or associations. Both persistent relationships and ephemeral relationships are possible, the difference being a matter of existence over time as related to the information being supported. Some relationships can be stored whereas many others are derived. In fact, there are many more opportunities for deriving relationships because storing relationships involves explicit management overhead. In early GIS, logical data models would represent topological (connectedness, adjacency and containment) structuring as a collection of pointers to records, whereas more recently topological relationships are derived by specifying rules for data classes. The storage of a relationship involves specifying a foreign key for an attribute, which means that an entity record from one entity class refers to an entity record within another entity class. Software operations establish a reference between the two entity records.

4. Schema Diagram of a Logical Data Model

As mentioned earlier, a logical data model extends the conceptual data model with more details, including primary keys (PK) and foreign keys (FK) (see below). A type hierarchy depicting inheritance of Tax Parcel and Development Parcel both inheriting characteristics, then adding characteristics as shown below (Figure 1).

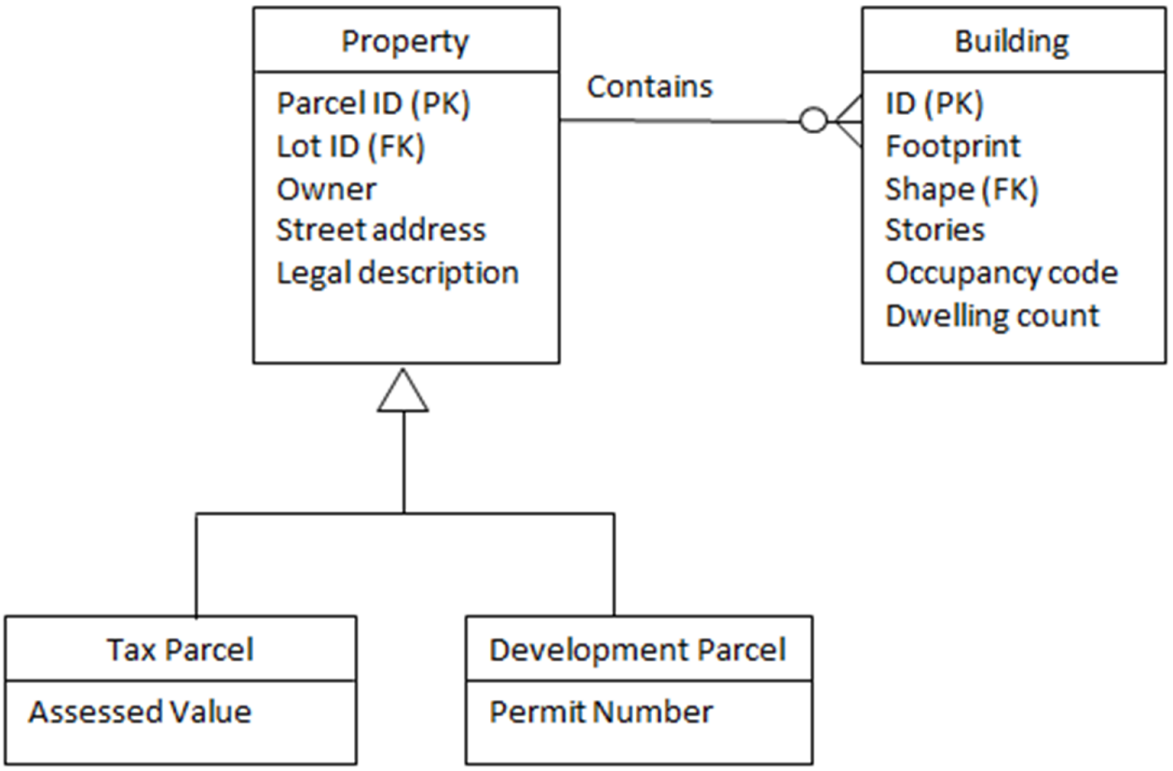

Figure 1. Entity-Relationship diagram with keys. Source: author.

5. Use Cases, Activity Diagrams, and Logical Data Models

Use cases and activity diagrams are among several artifacts developed by systems analysts for representing dynamics of information use. A use case contains a collection of steps that proceed from a pre-condition of the system to a post-condition of a system, thereby describing a task activity through time. A data modeler can develop a more detailed description of workflow from the use case diagram steps, translating each use case step into one or more activity diagrams (Rumbaugh, Booch, & Jacobsen, 1999). Each diagram can translate one step within the use case diagram into several more detailed steps to generate the information needed by a user. Thus, an activity diagram depicts the flow of work from one step to another, and would include the data input and output from each step. A logical data model can depict what data are needed by each step, and can be fleshed out by describing information use. Both the logical data model diagram and activity diagrams can be updated simultaneously assuming a data modeler and an end-user are working together on a team.

6. From Application-centric to Enterprise-wide Logical Data Models Using Schema Integration

Each application within a database system would have a logical data model schema diagram generated for it. An enterprise logical data model can be generated from the synthesis of application logical data models. This synthesis process is called schema integration (Nyerges, 1991). In schema integration, common entity types, attributes, and units of measurements for domains are compared among all entity types in a pair-wise manner. A decision is made to retain, reduce or eliminate the redundancy depending on the data management needs of the users. In some instances, leaving the redundancy in place can bolster faster access to data, hence increasing performance for data retrieval. Once entity types are processed, then relationship types are processed to determine whether they are to be collapsed or not. Schema integration and synthesis is an iterative process, using the conceptual schema to start but then inspecting the logical schemas to check if details are being missed or overlooked. Schema integration can be used to develop data warehouse and federated database applications (Yeung & Hall, 2007). A data warehouse supports decision support applications for strategic overview of organization concerns; whereas a federated database supports shared understanding about operational and mission critical application activities within an organization.

References

Chrisman, N. R. (2002). Exploring Geographic Information Systems, 2nd edition. New York, NY: Wiley.

Codd, E. F. (1980). Data models in database management. ACM SIGMOD Record - Proceedings of the workshop on Data abstraction, databases and conceptual modelling, 11(2), 112-114.

Martin, J. (1976). Principles of Data-Base Management. Englewood Cliffs, NJ: Prentice-Hall.

Nyerges, T. L. (1991). Schema Integration Analysis for GIS Database

Development. International Journal of Geographic Information Systems, 3(2), 153-183.

Rumbaugh, J., Jacobsen, I., & Booch, G. (1999). The Unified Modeling Language Reference Manual. Reading, MA: Addison-Wesley.

Simsion, G., & Witt, G. (2005). Data Modeling Essentials, 3rd edition. San Francisco, CA: Morgan Kaufmann Publishers Inc.

Stevens, S. S. (1946). On the Theory of Scales of Measurement. Science 103, 677-680.

West, M. (2011). Developing High Quality Data Models. San Francisco, CA: Morgan Kaufmann Publishers Inc.

Yeung, A. K. W. and Hall, G. B. (2007). Spatial Database Systems: Design, Implementation and Project Management, Springer, Dordrecht, Netherlands.