[DM-01-034] Conceptual Data Models

Abstract

Within an initial phase of database design, a conceptual data model is created as a technology-independent specification of the data to be stored within a database. This specification often times takes the form of a formalized diagram. The process of conceptual data modeling is meant to foster shared understanding among data modelers and stakeholders when creating the specification. As such, a conceptual data model should be easily readable by people with little or no technical-computer-based expertise because a comprehensive view of information is more important than a detailed view. In a conceptual data model, entity classes are categories of things (person, place, thing, etc.) that have attributes for describing the characteristics of the things. Relationships can exist between the entity classes. Entity-relationship diagrams have been and are likely to continue to be a popular way of characterizing entity classes, attributes and relationships. Various notations for diagrams have been used over the years. The main intent about a conceptual data model and its corresponding entity-relationship diagram is that they should highlight the content and meaning of data within stakeholder information contexts, while postponing the specification of logical structure to the second phase of database design called logical data modeling.

Keywords: database design

Author & citation

Nyerges, T. (2017). Conceptual Data Models. The Geographic Information Science & Technology Body of Knowledge (1st Quarter 2017 Edition), John P. Wilson (ed.). DOI: 10.22224/gistbok/2017.1.3

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). Conceptual model. The Geographic Information Science & Technology Body of Knowledge. Washington, DC: Association of American Geographers.

Explanation

1. Definitions
2. Data Models, Data Modeling, and Conceptual Data Modeling
3. Languages for Conceptual Data Modeling
4. Conceptual Data Model Diagramming
5. Use Cases and Conceptual Data Models
6. From Application-centric to Enterprise-wide Conceptual Data Models

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.

conceptual data model: A technology independent specification of the data to be held in a database. It is the focus of communication between business stakeholders and the data modeler. (Simsion & Witt, 2005, p. 17)

2. Data Models, Data Modeling, and Conceptual Data Modeling

From a data management perspective, two views of a data model are common; one view focuses on a data structure only, while second view includes a data structure plus operators and rules. The first view as represented by West (2011, p.5) defines a data model as '...the structure and intended meaning of data'. The more complete view given by Codd (1980, p. 112) defines a data model as three components: 1) a collection of data structure types (as described by West 2011); but also, 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. In the description of data model provided herein, we constrain the contribution to the popular understanding by West (2011), and recognize that Codd's (1980) view is particularly useful for designing database management software as opposed to data content and structure within a database design.

In a data management context, data modeling is a workflow process for performing database design that follows from semantic analysis of an application domain; wherein three levels of data model abstraction called conceptual, logical, and physical compose the full process of database design (Simsion & Witt, 2005). The contribution herein describes the conceptual data model phase of database design, whereas logical and physical data model phases are described in separate entries for the GIS&T Body of Knowledge. A conceptual data model is … "a (software) technology independent specification of the data to be held in a database" (Simsion & Witt, 2005, p. 17). A conceptual data model contains salient data content including relationships about a topic-centric domain for fostering shared understanding among database developers and stakeholders for an information system application. It is often simple in its content, but complex in its intent, because it may contain many-to-many relationships for a topic. A conceptual data model should be easily readable by people with little or no technical-computer-based expertise because a salient view of information is more important than a detailed view. Data meaning is conveyed by both the data classes as well as by relationships between/among classes. A conceptual data model is translated into a logical data model that emphasizes logical structure, which is then translated into a physical data model that emphasizes storage structures within particular data management software (See the GIS&T BOK contributions for logical data model and physical data model). The form of an expression for a conceptual data model could be a narrative or diagram, but it is commonly a diagram, and in a best

case, a machine readable diagram.

3. Languages for Conceptual Data Modeling

Peter Chen (1976) is credited with being among the first to formally characterize data at a conceptual level for implementation in database systems, and has become known as semantic data modeling. He called his approach, entity-relationship model. His ground-breaking approach invigorated a field of conceptual modeling for artificial intelligence, databases, and programming languages (Brodie, Mylopoulos, & Schmidt, 1984). Up until the late 1990's, the entity-relationship (ER) approach continued to be among the most popular conceptual data modeling approaches for information engineering. In late 1990's the object-role model (ORM) appeared (Halpin, 1998) as an alternative to the Entity-Relationship Approach. In ORM an attribute for an object (object or variable) is translated into a relationship that is called a role for the object and is said to be more easily understood by users. In addition, in the late 1990's object-orientation and object modeling techniques were merged into the unified modeling language (UML) as an attempt to generalize software systems engineering (Rumbaugh, Jacobsen, & Booch, 1999). Several types of conceptual data model diagrams more generally called 'structure diagrams' now exist as part of object modeling techniques. UML object class diagrams have been used for database design, particularly at the logical and physical levels (Naiburg & Maksimchuk, 2001). Halpin (2002) provides a comparison among ER, ORM, and UML approaches to conceptual modeling. There are more similarities than differences, but one must choose a language to express the semantics of an application. Both ER and ORM have been used predominantly for database modeling. UML has been used more often for software engineering. ER conceptual data models have simple constructs for representing information, with few rules for fostering meaningful expression. Constructs commonly include entity class, attribute and relationships which are notated in a schema document. Consequently, we describe the ER approach here, but encourage the reader to explore the other approaches provided through the reference citations.

3.1 Entity and Entity Class

An entity is a person, place, or thing of some past, present, future or imaginary world for which measurements can be taken using one or more attributes. An entity class is the collection of entities with similar attributes. An entity type can be considered the definition of the class. One of the biggest problems in data management involves identifying, naming, and defining entity classes. To share insight we need to share understanding about a topic. Making context an explicit part of the data, i.e., describe all the data needed for context, fosters data clarity. Entity classes should represent, and be named after, the underlying attributes of a person, place or thing, not the role it plays in a particular context. A role is a secondary attribute, and not a primary attribute of entity (Simsion & Witt, 2005).

3.2 Entity Attributes

The underlying character of a person, place or thing is specified in terms of attributes. Primary and secondary attributes can be identified as the basis of classifying and defining an entity type (Nyerges, 1991). A primary attribute is an attribute that must be included as part of the class type for characterizing its identity. A secondary attribute is an attribute

that is not necessarily needed for specification of a class type. An entity type is formed by bundling a collection of primary attributes to give an entity class meaning; together these bundled attributes form the identity of the entity type. When one removes one of the primary attributes then the entity type becomes a different class by identity of the type. Following from the primary attribute inclusion above, when a secondary attribute is removed from an entity type, the entity type remains the same class. Candidate attributes should be treated as representing relationships to other entity types. Every attribute is a candidate for a relationship to another entity (Simsion & Witt, 2005).

3.3 Relationships

An entity class defined by an entity type can be associated with another entity class or to itself through a relationship class. Relationships that persist over time are called persistent or structural relationships. Temporary relationships can exist and these are called ephemeral or incidental relationships. Persistent relationships are more easily designed, than those that are ephemeral. Relationships often arise due to the particular type and level of attributes qualifying the entities. Activities and associations should be represented by entity types, as relationships provide insight to the roles involved with an entity.

Relationships between entities have cardinality. Cardinality refers to the number of instances of one entity class related to instances of another entity class. There potentially exist, 0, 1, or more instances of one entity class that could be related to instance(s) of another entity class. Some designers view cardinality as a logical data model issue rather than a conceptual data model issue (West, 2011). If cardinality is not represented at the conceptual level, then it should be represented at the logical data model level. The difference in approach is dependent on whether collections of entities are meaningful for interpretation of application problems. They often are for geospatial problems, so representing them at the conceptual level is preferred.

4. Conceptual Data Model Diagramming

A schema is assumed to be machine readable, and often takes the form of a diagram to enhance readability. Various notations in line with different languages have emerged for formulating conceptual data models as described earlier. Below we first discuss notation associated with ER modeling and then turn to an example about land parcels.

4.1 Diagram Notation

Diagrams are commonly created to enhance understanding of complex topics. Because databases often contain considerable information, diagramming becomes an important supplement to the design process. An entity-relationship diagram is composed of three basic components as mentioned above, and thus requires three notation categories, one for each of the entity (class) types, attributes, and relationships. Standard notation for entity types has commonly involved the use rectangular boxes with a label for the entity type name. Sometimes the label is outside the box, sometimes inside the box; it really depends on the software vendor, and what notation they have adopted. Consistency is really the rule to be used, regardless of where the label might appear. Standard notation for attributes has involved both the listing of attributes within box as well as at the side of a box. Sometimes notation has been in a separate box to the side, but that is less common.

A variety of notations for relationships have emerged over the years, many of the differences associated with how to best characterize relationships, and particularly cardinality for relationships, as there are many ways to depict this information. A triangle with lines emanating from either side labeled with the cardinality was an original symbol used in ER Notation (Chen 1976). Lines with arrowheads on either end of the line, together with a numeric designation for cardinality, have been used in UML. Lines with crows-feet representation, wherein a single line meant 1 and three lines meant many, with a 0 designating none was used. Simsion and Witt (2005, p. 83) show the same relationship depicted using six different software tools, each having slightly different notation. There is no single standard considered 'best', but of course some software products are more widely distributed than others.

4.2 An Example of Conceptual Data Model Entity, Attribute, and Relationship Diagram

Below is a modified ER Diagram that depicts a property entity class and a building entity class (Figure 1). A property has attributes: parcel identifier, lot identifier, and owner. A building entity class has attributes: footprint and number of stories. There exists a relationship between them which is 'contains'. A property contains a building and a building is contained by a property. The cardinality is read, every property might have 0, 1, or more buildings contained within its boundary. The crows-feet representation is used to designate the 0 or 1-to-many cardinality.

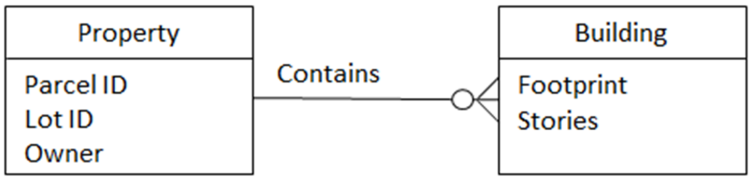


Figure 1. Entity-Relationship Diagram

5. Use Cases and Conceptual Data Models

A use case characterizes a sequence of steps as part of business process modeling when examining an application task. A use case results from undertaking business process modeling to identify the conditions before the sequence of steps (pre-conditions), the steps themselves, and the conditions following the sequence of steps (post-conditions). Two levels of use cases can be articulated, business use case and system use case. Business use cases focus on the functionality of information system tasks, whereas system use cases focus on task execution. Since a conceptual data model represents a high-level view of information requirements, the business use case is more applicable to the development of data classes for conceptual data models. In a business use case, activity sequence leads to an understanding of the types of data needed. Data classes can be derived from the nouns within the task descriptions. In a system use case, however, a conceptual data model could be used as part of the pre-conditions of the tasks to be undertaken. That is, a conceptual data model would exist prior to the execution of task, and is a data framework that could be used to execute the task to further articulate data classes and relationships needed between them.

6. From Application-centric to Enterprise-wide Conceptual Data Models

The above discussion focused on a single application within an information system. However, organizations often undertake many applications, with a high likelihood that many applications share data requirements. Consequently, it is advantageous to assemble application-centric conceptual data models, synthesizing among them to create an enterprise-wide conceptual data model. The main purpose of creating an enterprise conceptual data model is that it fosters shared understanding among applications, while reducing inconsistency in data and information provided to users.

From the results, an integrated schema is developed that can be used to support a data warehouse or federated database approach (Yeung & Hall, 2007). Both data warehouse and federated database environments foster shared access to data, and thus both a multi-user database environment. However, their database missions differ. A data warehouse commonly stores data for decision support applications wherein the data schema will not change substantially, but the data will change over time. A federated database is a collection of schemas developed with the intent to share data among the schemas commonly more operational, e.g. for mission critical day-to-day activities within an organization.

References

Brodie, M. L., Mylopoulos, J., & Schmidt, J. W. (1984). On Conceptual Modeling. New York, NY: Springer-Verlag.

Chen, P. (1976). The Entity–Relationship Model – Toward a Unified View of Data. ACM Transactions on Database Systems. New York: Association for Computing Machinery. 1(1): 9–36.

Codd, E. F. (1970). A Relational Data Model for Large Shared Data Banks. Communications of the ACM. 13(6), 377-387.

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.

Halpin, T. (1998). ORM/NIAM Object-Role Modeling. In: Bernus, P., Mertins, K., Schmidt, G. (eds) Handbook on Architectures of Information Systems. International Handbooks on Information Systems. Springer, Berlin, Heidelberg.

Halpin, T. (2002). Metaschemas for ER, ORM and UML Data Models: A Comparison. Journal of Database Management, 13(2), 20-30.

Naiburg, E. J., & Maksimchuk, R. A. (2001). UML for Database Design, 1st Edition. Reading, MA: Addison-Wesley.

Nyerges, T. L. (1991). Geographic Information Abstractions: Conceptual Clarity for Geographic Modeling. Environment and Planning A, 23, 1483-1499.

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.

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.