[DM-01-036] Physical Data Models

Abstract

Constructs within a particular implementation of database management software guide the development of a physical data model, which is a product of a physical database design process. A physical data model documents how data are to be stored and accessed on storage media of computer hardware. A physical data model is dependent on specific data types and indexing mechanisms used within database management system software. Data types such as integers, reals, character strings, plus many others can lead to different storage structures. Indexing mechanisms such as region-trees and hash functions and others lead to differences in access performance. Physical data modeling choices about data types and indexing mechanisms related to storage structures refine details of a physical database design. Data types associated with field, record and file storage structures together with the access mechanisms to those structures foster (constrain) performance of a database design. Since all software runs using an operating system, field, record, and file storage structures must be translated into operating system constructs to be implemented. As such, all storage structures are contingent on the operating system and particular hardware that host data management software.

Keywords: database design

Author & citation

Nyerges, T. (2017). Physical Data Models. The Geographic Information Science & Technology Body of Knowledge (1st Quarter 2017 Edition), John P. Wilson (ed.).
DOI: 10.22224/gistbok/2017.1.1

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). Physical models. The Geographic Information Science & Technology Body of Knowledge. Washington, DC: Association of American Geographers. (2nd Quarter 2016, first digital)

Explanation

1. Definitions
2. Data Models, Data Modeling, and Physical Data Modeling
3. Components of a Physical Data Model
4. Schema Diagram of a Physical Data Model
5. Database Management Systems

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.

physical data model: A translation of the logical model into physical storage structures and access mechanisms to achieve performance within particular database management software. (Simsion & Witt, 2005, p. 17)

2. Data Models, Data Modeling, and Physical Data Modeling

One of the most comprehensive definitions of a data model was provided by Edgar Codd (1980) ten plus years after he developed the relational data model (Codd 1970). Codd's interest stemmed from clarifying the logical character of a data model, as opposed to its physical implementation. As in the characterization of conceptual and logical data models found in separate entries of the GIS&T Body of Knowledge, from a database design perspective, a more common and popular understanding of data model is that it defines the structure and intended meaning of data (West 2011, p. 5). However, Codd's (1980, p. 112) more detailed view characterizes a data model as consisting of three components: 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 physical data models we emphasize the contribution to the view given by Martin (1976) and West (2011). However, we point out that Codd's (1980) definition provides additional insight particularly useful for understanding data management software design in addition to database design.

In alignment with use of data models, data modeling describes a database design process composed of creating conceptual, logical and physical data models for the purpose of creating a fully implementable database design. A physical data model is developed for the third of three levels based on a logical data model design, and it is this physical data model that is the topic of this entry with in the GIS&T Body of Knowledge. A physical data model further characterizes the descriptions of the logical data model constructs in terms of implementation relevant to specific data management software. It adds information about storage structures plus data access details that enables and/or constrains performance. A physical data model is an operational model to characterize the database working within the software and hardware environment chosen for implementation.

3. Components of a Physical Data Model

All database management system (DBMS) software packages embed a physical data model within them as an extension of a logical data model, and hence the formalism upon which

the logical data model is based. The GIS&T BoK entry for logical data model describes several types of logical data models, e.g. relational, graph, object-oriented. With this perspective in mind, it is clear why data management software are enabled and constrained by the logical model that guides the capabilities available. Data logical structures tables, objects, attribute fields and relationships are implemented as physical data storage structures with data access mechanisms for primary and foreign keys. Logical operations and their physical implementation are used to derive logical structures and store them in terms of storage structures. Integrity rules constrain structures and operations in both the logical and physical sense.

In the 1990's, many GIS software designers downplayed the simple but robust structures of the relational model for logical data organization (Codd 1970). Relational tables with columns for attribute fields and rows with instances for the entities foster duplicity of geospatial data that characterize logical relationships, e.g. topological relationships. More recently, to avoid duplicity within a geospatial context, relational data model developers have extended the data types to include abstract data types sufficiently expressive of geospatial structures. Furthermore, other software, called object stores, is used to establish logical spatial structure for advanced applications, adding to the extended structuring capability of the relational model. Esri's geodatabase data model uses this strategy, wherein the relational model is used to store physical level data coordinate instances. That change in approach has come with the advent of software 'tiering' design which has introduced representation flexibility at the logical level, while preserving robust implementation structures, particularly when enterprise data management is needed to support multi-user database environments.

As mentioned earlier, differences among DBMS software within a GIS context can be attributed to the difference in logical model in terms of logical spatial data types (data structures), the corresponding operations, and the rules that are implemented to maintain integrity for generating information. However, even within types, additional differences can exist among the way in which each software system implements the basic data types, e.g., integers, reals, and character strings. It is these differences that make the databases physically different. For example, even in the Esri spatial data engine (multi-user) geodatabase environment, wherein multiple DBMS platforms can implement a geodatabase, the geodatabases although similar at the logical level, are different at the physical level because the typing is implemented differently. Furthermore, the many logical tables within a relational model are often stored within a single (or very few) physical (operating system) files to speed the performance of an implementation. However, they each do it their own way which makes the implementation special to the particular software. These physical file formatting differences can enable different levels of performance for the same logical representation of data.

A physical data model extends and deepens logical data structuring with physical storage structures related to fields, records and files implemented within software. Since all software runs on top of an operating system, the fields, records, and files must be translated to the operating system level to be implemented. As such, all storage structures are contingent on the operating system and particular hardware that host the data management software.

3.1 Data Types for Data Fields

A physical data model uses well-defined data type formats, which are often unique to a DBMS software implementation, although not entirely different among software implementations. For example, a list of the data types implemented in the PostgreSQL database management system are listed in the table below.

Table 1. Selected PostgreSQL Data Types (Adapted from: https://www.techonthenet.com/postgresql/datatypes.php)

bit(size)	Fixed-length bit string Where **size** is the length of the bit string.
varbit(size) bit varying(size)	Variable-length bit string Where **size** is the length of the bit string.
Smallint, Int, Bigint	Equivalent to int2, int4, or int8, as .2-byte, 4-byte or 8-byte signed integer, respectively.
Smallserial, Serial, Bigserial	Auto-incrementing integer value which is equivalent to serial2, serial4 or serial 8, as 2-byte, 4-byte or 8-byte signed integer, respectively. 2-byte signed integer that is auto-incrementing.
numeric(m,d)	Where **m** is the total digits and **d** is the number of digits after the decimal.
Real, Double Precision	4-byte, single precision, floating-point number; 8-byte, double precision, floating-point number, respectively
Money	Currency value.
Bool, Boolean	Logical boolean data type - true or false

3.2 Spatial Indexing

Indexing, particularly spatial indexing is a major part of the physical data model design of geospatial databases because a common user task involves spatial querying. Among the most popular of spatial indexes are the quadtree, R-Tree, B-Tree and Hash. A quadtree is most useful for tiled surfaces because of the regularity of the tiles. The spatial domain is decomposed into quadrants, and each quadrant is further sub-divided into quadrants. With iteration a tree is formed for rapid access to the tiles. For vector objects in a space, the R-Tree is used. Each spatial object (e.g., polygon) has a minimum enclosing rectangle (MER) that encompasses the boundary of polygon. MERs can be aggregated to summarize regions of spatial objects, and are added. Objects are added to an MER within the index that will lead to the smallest increase in its size. A B-Tree has been useful for single dimensional work; a single dimension offering a range of data values, e.g, parcel size, over which the index is developed. A Hash-Table has been useful for computing foreign keys.

- Quadtree: often a preferred method for indexing spatial tiles
- R-tree: often a preferred method for indexing spatial data objects, e.g. polygons. lines and points, are grouped using the MER.
- B-tree: a self-balancing tree data structure supporting a data sort with searches, sequential access, insertions, and deletions in logarithmic time
- Hash function: computes an index of buckets or slots in which desired value can be found

4. Schema Diagram of a Physical Data Model

The schema diagram below includes that information using the data type formats from PostgreSQL software.

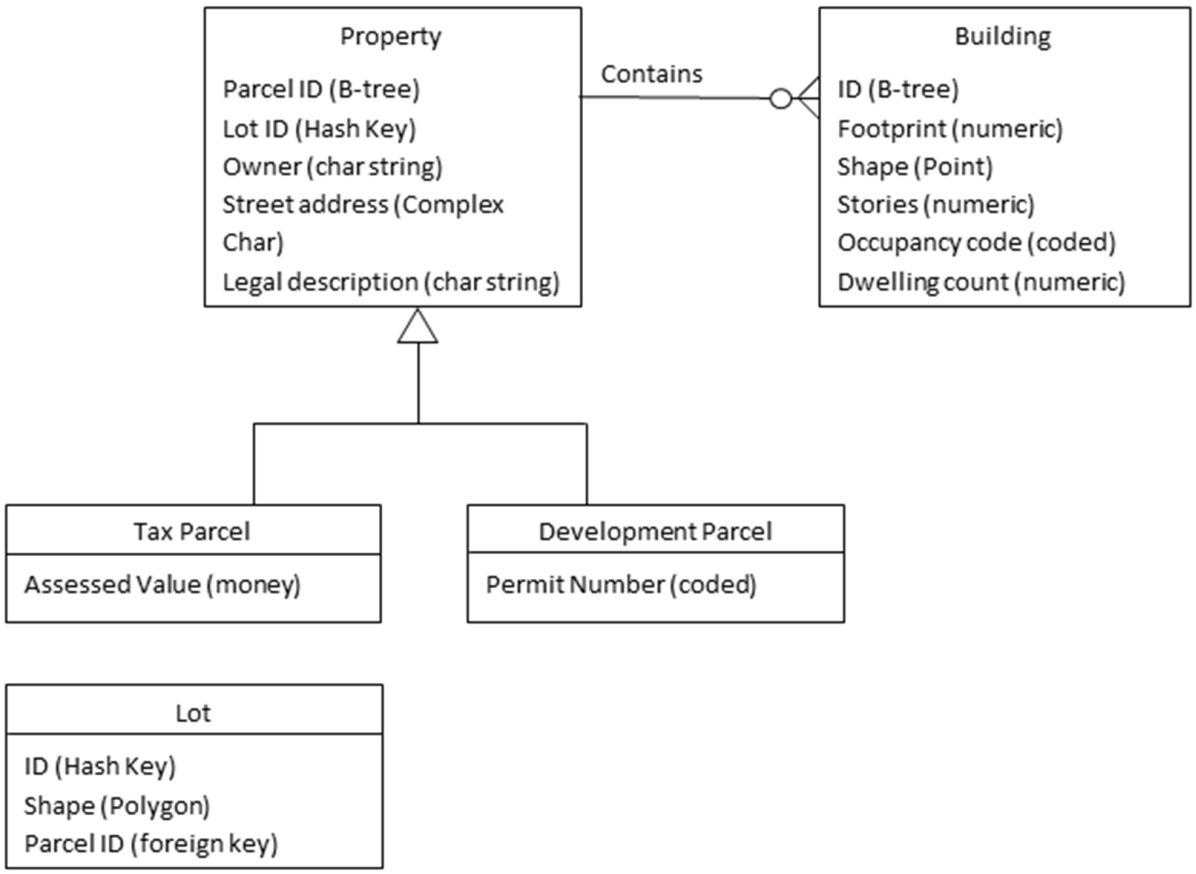


Figure 1. Entity-Relationship diagram for physical data model

The relationships between entities are commonly implemented with record pointers, often called a 'foreign key' (Shekhar & Chawla, 2003). Foreign keys are used to address from one record to another, implementing access across the records. Figure 1 has a foreign key from lot to property parcel.

5. Database Management Systems

A variety of approaches (logical data model types) for implementing DBMS exist, each type being a different implementation of a logical data language. (See the GIS&T BoK entry for logical data model description.) The Wikipedia page provides a wide-assortment of DBMS implementations, and here they have been categorized by logical data model approach. The original list has been edited.

Relational

- Caliper extends the Raima Data Manager with spatial datatypes, functions, and utilities

- IBM DB2 Spatial Extender can spatially-enable any edition of DB2, including the free DB2 Express-C, with support for spatial types
- SpatiaLite extends Sqlite with spatial datatypes, functions, and utilities
- Oracle Spatial
- Microsoft SQL Server has support for spatial types since version 2008
- PostgreSQL DBMS uses the spatial extension PostGIS to implement the standardized datatype geometry and corresponding functions
- Esri File geodatabase, plus support of single-user and multiuser relational geodatabases
- CartoDB, a cloud-based geospatial database on top of PostgreSQL with PostGIS
- H2 supports geometry types and spatial indices as of version 1.3.173 (2013-07-28); an extension called H2GIS available on Maven Central gives full OGC Simple Features support
- Linter SQL Server supports spatial types and spatial functions according to the OpenGIS specifications

Relational Column Store

- Vertica Place, the geo-spatial extension for HP Vertica, adds OGC-compliant spatial features to the relational column-store database
- MonetDB/GIS extension for MonetDB adds OGC Simple Features to the relational column-store database

Object-oriented

- Smallworld VMDS, the native GE Smallworld GIS database

Graph

- Neo4j, a graph database that can build 1D and 2D indexes as B-tree, Quadtree and Hilbert curve directly in the graph
- AllegroGraph is a graph database which provides a novel mechanism for efficient storage and retrieval of two-dimensional geospatial coordinates for Resource Description Framework data; it includes an extension syntax for SPARQL queries.

NoSQL Document Store

- MarkLogic, MongoDB, and RethinkDB support geospatial indexes in 2D
- GeoMesa is a cloud-based spatio-temporal database built on top of Apache Accumulo and Apache Hadoop; GeoMesa supports full OGC Simple Features and a GeoServer plugin
- RavenDB supports geospatial indexes in 2D
- CouchDB a document-based database system that can be spatially enabled by a plugin called Geocouch

NoSQL Key-value Store

- Redis with the Geo API
- Tarantool supports geospatial queries with RTREE index

Open Standards

- SpatialDB by MineRP, an open-standards spatial database with spatial type extensions used mostly within the mining industry

The majority of DBMS available within the above list have been implemented based on the relational data model, or a derivation thereof, due to its long history of success. However, there are many other DBMS implementations based on other logical data models as well. There is no implied recommendation in the listing.

A general (using six criteria) ranking of popularity among 300+ DBMS is maintained on a website page by DB-Engines (2016). The top-ten ranked DBMS are the following.

Table 2. Top-ten ranked DBMS by DB-Engines as of November 30, 2016

Rank	DBMS Name	Data Model
1.	Oracle	Relational
2.	MySQL	Relational
3.	Microsoft SQL Server	Relational
4.	PostgreSQL	Relational
5.	MongoDB	NoSQL Document store
6.	DB2	Relational
7.	Cassandra	Wide column store
8.	Microsoft Access	Relational
9.	Redis	NoSQL Key-value store
10.	SQLite	Relational

Unfortunately, no ranked DBMS list for support of the Esri geodatabase data model, for the ArcGIS Enterprise approach, has been found in the literature. Such a ranking might provide an interesting glimpse of how GIS installations across the world are making use of enterprise DBMS solutions. Nonetheless, the trend is clear that more organizations are adopting GIS enterprise solutions as the array of spatial data approaches continue to expand with the implementation and use of GIS.