

Data Models

Hi! In this lecture you are going to learn about the data models used for designing a database.

Before the data available in an enterprise can be put in a DBMS, an overall abstract view of the enterprise data must be developed. The view can then be translated into a form that is acceptable by the DBMS. Although at first sight the task may appear trivial, it is often a very complex process. The complexity of mapping the enterprise data to a database management system can be reduced by dividing the process into two phases. The first phase as noted above involves building an overall view (or model) of the *real world* which is the enterprise (often called the *logical database design*). The objective of the model is to represent, as accurately as possible, the information structures of the enterprise that are of interest. This model is sometimes called the *enterprise conceptual schema* and the process of developing the model may be considered as the requirements analysis of the database development life cycle. This model is then mapped in the second phase to the user schema appropriate for the database system to be used. The logical design process is an abstraction process which captures the meaning of the enterprise data without getting involved in the details of individual data values. Figure 2.1 shows this two step process.

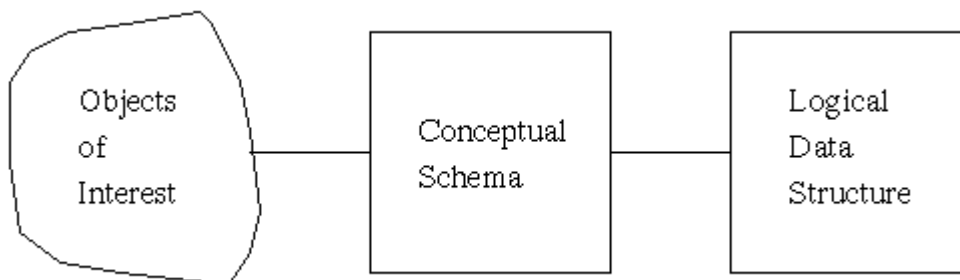


Figure 2.1

The process is somewhat similar to modeling in the physical sciences. In logical database design, similar to modeling in the physical sciences, a model is built to enhance understanding and abstracting details. A model cannot be expected to provide complete

knowledge (except for very simple situations) but a good model should provide a reasonable interpretation of the real-life situation. For example, a model of employee data in an enterprise may not be able to capture the knowledge that employees May Adams and John Adams are married to each other. We accept such imperfections of the model since we want to keep the model as simple as possible. It is clearly impossible (and possibly undesirable) to record every available piece of information that is available in an enterprise. We only desire that the meaning captured by a data model should be adequate for the purpose that the data is to be used for.

The person organizing the data to set up the database not only has to model the enterprise but also has to consider the efficiency and constraints of the DBMS although the two-phase process separates those two tasks. Nevertheless, we only consider data models that can be implemented on computers.

When the database is complex, the logical database design phase may be very difficult. A number of techniques are available for modeling data but we will discuss only one such technique that is, we believe, easy to understand. The technique uses a convenient representation that enables the designer to view the data from the point of view of the whole enterprise. This well known technique is called the *entity-relationship model*. We discuss it now.

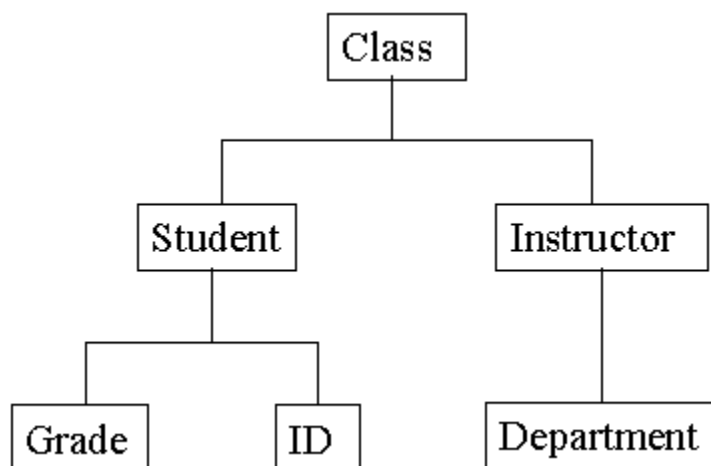
TYPES OF DATABASE MANAGEMENT SYSTEMS

A DBMS can take any one of the several approaches to manage data. Each approach constitutes a database model. A data model is a collection of descriptions of data structures and their contained fields, together with the operations or functions that manipulate them. A data model is a comprehensive scheme for describing how data is to be represented for manipulation by humans or computer programs. A thorough representation details the types of data, the topological arrangements of data, spatial and temporal maps onto which data can be projected, and the operations and structures that can be invoked to handle data and its maps. The various Database Models are the following:-

- *Relational* – data model based on tables.
- *Network* – data model based on graphs with records as nodes and relationships between records as edges.
- *Hierarchical* – data model based on trees.
- *Object-Oriented* – data model based on the object-oriented programming paradigm.

Hierarchical Model

In a Hierarchical model you could create links between these record types; the hierarchical model uses Parent Child Relationships. These are a 1: N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from maths. For example, an organization might store information about an employee, such as name, employee number, department, salary. The organization might also store information about an employee's children, such as name and date of birth. The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment. If an employee has three children, then there would be three child segments associated with one employee segment. In a hierarchical database the parent-child relationship is one to many. This restricts a child segment to having only one parent segment. Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s.



Advantages

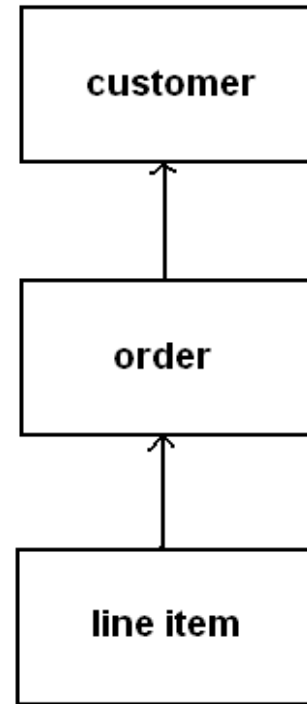
- Simplicity
- Data Security and Data Integrity
- Efficiency

Disadvantages

- Implementation Complexity
- Lack of structural independence
- Programming complexity

The figure on the right hand side is a Customer-order-line item database:

There are three data types (record types) in the database: customers, orders, and line items. For each customer, there may be several orders, but for each order, there is just one customer. Likewise, for each order, there may be many line items, but each line item occurs in just one order. (This is the schema for the database.) So, each customer record is the root of a tree, with the orders as children. The children of the orders are the line items. Note: Instead of keeping separate files of Customers, Orders, and Line Items, the DBMS can store orders immediately after customers. If this is done, it can result in very efficient processing.



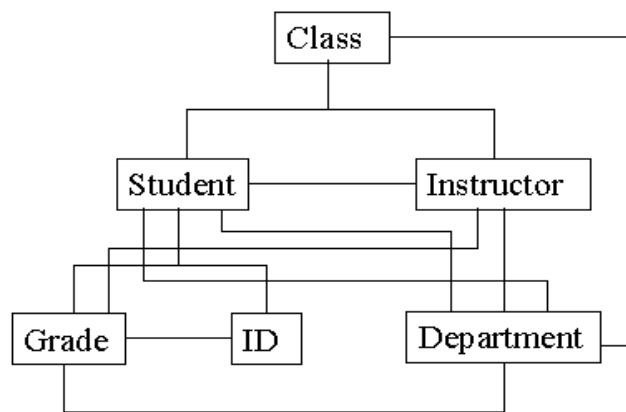
Problem: What if we also want to maintain Product information in the database, and keep track of the orders for each product?

Now there is a relationship between orders and line items (each of which refers to a single product), and between products and line items. We no longer have a tree structure, but a directed graph, in which a node can have more than one parent.

In a hierarchical DBMS, this problem is solved by introducing pointers. All line items for a given product can be linked on a linked list. Line items become "logical children" of products. In an IMS database, there may be logical child pointers, parent pointers, and physical child pointers

NETWORK DATA MODEL

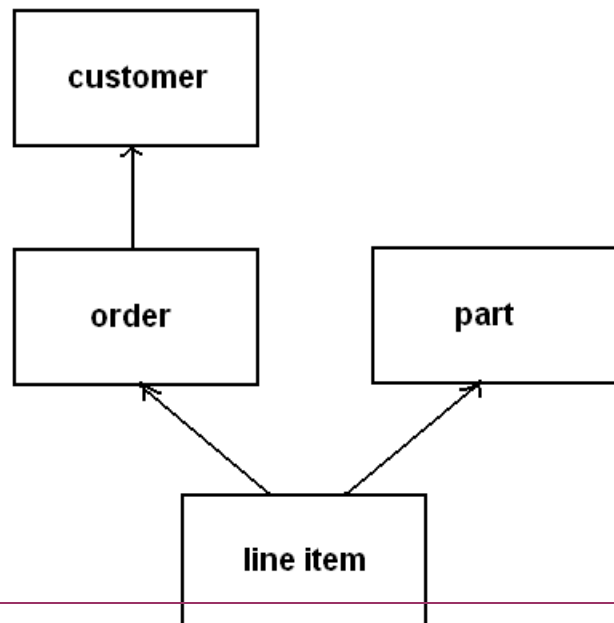
A member record type in the Network Model can have that role in more than one set; hence the multivalent concept is supported. An owner record type can also be a member or owner in another set. The data model is a simple network, and link and intersection record types (called junction records by IDMS) may exist, as well as sets between them. Thus, the complete network of relationships is represented by several pair wise sets; in each set some (one) record type is owner (at the tail of the network arrow) and one or more record types are members (at the head of the relationship arrow). Usually, a set defines a 1:M relationship, although 1:1 is permitted. The CODASYL network model is based on mathematical set theory.



NETWORK DATA MODEL

Advantages

- Conceptual Simplicity
- Ease of data access
- Data Integrity and capability to handle more relationship types
- Data independence
- Database standards
- Disadvantages
- System complexity



- Absence of structural independence

Instead of trees, schemas may be acyclic directed graphs.

In the network model, there are two main abstractions: *records* (record types) and *sets*. A set represents a one-to-many relationship between record types. The database diagrammed above would be implemented using four records (customer, order, part, and line item) and three sets (customer-order, order-line item, and part-line item). This would be written in a *schema* for the database in the network DDL.

Network database systems use linked lists to represent one-to-many relationships. For example, if a customer has several orders, then the customer record will contain a pointer to the head of a linked list containing all of those orders.

The network model allows any number of one-to-many relationships to be represented, but there is still a problem with many-to-many relationships. Consider, for example, a database of students and courses. Each student may be taking several courses. Each course enrolls many students. So the linked list method of implementation breaks down (Why?)

The way this is handled in the network model is to decompose the many-to-many relationship into two one-to-many relationships by introducing an additional record type called an "intersection record". In this case, we would have one intersection record for each instance of a student enrolled in a course.

This gives a somewhat better tool for designing databases. The database can be designed by creating a diagram showing all the record types and the relationships between them. If necessary, intersection record types may be added. (In the hierarchical model, the designer must explicitly indicate the extra pointer fields needed to represent "out of tree" relationships.)

In general, these products were very successful, and were considered the state of the art throughout the 1970s and 1980s. They are still in use today.

- IMS (hierarchical) IBM
- IDMS (network) Computer Associates
- CODASYL DBMS (network) Oracle

But -- there are still some problems.

There is an insufficient level of data abstraction. Database designers and programmers must still be cognizant of the underlying physical structure.

Pointers are embedded in the records themselves. That makes it more difficult to change the logical structure of the database. Processing is "one record at a time". Application programs must "navigate" their way through the database. This leads to complexity in the applications. The result is inflexibility and difficulty of use.

Performing a query to extract information from a database requires writing a new application program. There is no user-oriented query language. Because of the embedded pointers, modifying a schema requires modification of the physical structure of the database, which means rebuilding the database, which is costly.

Relational Model

A database model that organizes data logically in tables. A formal theory of data consisting of three major components: (a) A structural aspect, meaning that data in the database is perceived as tables, and only tables, (b) An integrity aspect, meaning that those tables satisfy certain integrity constraints, and (c) A manipulative aspect, meaning that the tables can be operated upon by means of operators which derive tables from tables. Here each table corresponds to an application entity and each row represents an instance of that entity. (RDBMS - relational database management system) A database based on the relational model was developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of Relational Tables:

- Values Are Atomic
- Each Row is Unique
- Column Values Are of the Same Kind
- The Sequence of Columns is Insignificant
- The Sequence of Rows is Insignificant
- Each Column Has a Unique Name

Certain fields may be designated as keys, which mean that searches for specific values of that field will use indexing to speed them up. Often, but not always, the fields will have the same name in both tables. For example, an "orders" table might contain (customer-ID, product-code) pairs and a "products" table might contain (product-code, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables. This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retrieval time, relational databases are classed as dynamic database management system. The RELATIONAL database model is based on the Relational Algebra.

Advantages

- Structural Independence
- Conceptual Simplicity
- Ease of design, implementation, maintenance and usage.
- Ad hoc query capability
- Disadvantages
- Hardware Overheads
- Ease of design can lead to bad design

The relational model is the most important in today's world, so we will spend most of our time studying it. Some people today question whether the relational model is not too simple, that it is insufficiently rich to express complex data types.

Note: Database theory evolved from file processing. So ideas that were driving programming languages and software engineering were not part of it.

Unit-1 Database System Concept Lecture-3

Example A table of data showing this semester's computer science courses

course number	section number	title	room	time	instructor
150	01	Principles of Computer Science	King 221	MWF 10-10:50	Bilar
150	02	Principles of Computer Science	King 135	T 1:30-4:30	Geitz
151	01	Principles of Computer Science	King 243	MWF 9-9:50	Bilar
151	02	Principles of Computer Science	King 201	M 1:30-4:30	Bilar
151	03	Principles of Computer Science	King 201	T 1:30-4:30	Donaldson
210	01	Computer Organization	King 221	MWF 3:30-4:20	Donaldson
280	01	Abstractions and Data Structures	King 221	MWF 11-11:50	Geitz
299	01	Mind and Machine	King 235	W 7-9:30	Borrioni
311	01	Database Systems	King 106	MWF 10-10:50	Donaldson
383	01	Theory of Computer Science	King 227	MWF 2:30-3:30	Geitz

Codd's idea was that this is the way that humans normally think of data. It's simpler and more natural than pointer-based hierarchies or networks. But is it as expressive?

Object Oriented Data Models

Object DBMSs add database functionality to object programming languages. They bring much more than persistent storage of programming language objects. Object DBMSs extend the semantics of the C++, Smalltalk and Java object programming languages to provide full-featured database programming capability, while retaining native language compatibility. A major benefit of this approach is the unification of the application and database development into a seamless data model and language environment. As a result, applications require less code, use more natural data modeling, and code bases are easier to maintain. Object developers can write complete database applications with a modest amount of additional effort.

In contrast to a relational DBMS where a complex data structure must be flattened out to fit into tables or joined together from those tables to form the in-memory structure, object DBMSs have no performance overhead to store or retrieve a web or hierarchy of interrelated objects. This one-to-one mapping of object programming language objects to database objects has two benefits over other storage approaches: it provides higher performance management of objects, and it enables better management of the complex interrelationships between objects. This makes object DBMSs better suited to support applications such as financial portfolio risk analysis systems, telecommunications service applications, World Wide Web document structures, design and manufacturing systems, and hospital patient record systems, which have complex relationships between data.

Advantages

- Capability to handle large number of different data types
- Marriage of object-oriented programming and database technology
- Data access
- Disadvantages
- Difficult to maintain
- Not suited for all applications

Some current and future trends

1. Object-oriented databases.

Can we apply object-oriented theory to databases? Some object-oriented commercial systems have been developed, among them O2, Objectivity, and POET. None of these is dominant or particularly successful. There is no standard for O-O databases. One trend is toward "object-relational" systems (like Oracle 8i, 9i). Oracle has added object-oriented features to their existing relational system.

2. Multimedia data.

Traditional DBMSs handled records with fields that were numbers or character strings, a few hundred characters long at most. Now, DBMSs must handle picture and sound files, HTML documents, etc.

3. Data warehousing and data mining.

Maintain storage of large amounts of data, often historical or on legacy systems, to support planning and analysis. Searching for trends and other information in existing data.

Bottom line: We expect a DBMS to support:

1. Data definition. (schema)
2. Data manipulation. (queries and updates)
3. Persistence. (long-term storage)
4. Multi-user access.

Summary

A DBMS can take any one of the several approaches to manage data. Each approach constitutes a database model

The various Database Models are the following:-

Relational - data model based on tables.

Network - data model based on graphs with records as nodes and relationships between records as edges.

Hierarchical - data model based on trees. Object-Oriented - data model based on the object-oriented programming paradigm

Questions

1. Explain the network data model?
2. Explain the Hierarchical data model?
3. Explain relational data model?
4. Explain the new trends in DBMS

References

<http://www.tc.cornell.edu/services/edu>

<http://www.unixspace.com/context/databases.html>

Date, C.J., Introduction to Database Systems (7th Edition) Addison Wesley, 2000

Leon, Alexis and Leon, Mathews, Database Management Systems, LeonTECHWorld.