

UC Santa Barbara

Core Curriculum-Geographic Information Science (1997-2000)

Title

Unit 045 - Non-Spatial Database Models

Permalink

<https://escholarship.org/uc/item/3dj9q8m4>

Authors

045, CC in GIScience
Meyer, Thomas H.

Publication Date

2000

Peer reviewed

Unit 045 - Non-Spatial Database Models

by Thomas H. Meyer, Mapping Sciences Laboratory, Texas A&M University, USA

This unit is part of the *NCGIA Core Curriculum in Geographic Information Science*. These materials may be used for study, research, and education, but please credit the author, Thomas H. Meyer, and the project, *NCGIA Core Curriculum in GIScience*. All commercial rights reserved. Copyright 1997 by Thomas H. Meyer.

Advanced Organizer

Topics covered in this unit

- This unit introduces the terms and concepts needed to understand non-spatial databases and their underlying data models, including:
 - a motivation of the need for database management systems
 - an overview of database terminology
 - a description of non-spatial data models

Intended Learning Outcomes

- After learning the material covered in this unit, students should be able to:
 - explain the purpose of a database management system
 - list the major non-spatial data models and their features
 - identify the primary distinctions between the major non-spatial data models

[Instructors' Notes](#)

[Full Table of Contents](#)

[Metadata and Revision History](#)

Non-spatial Database Models

1. Motivation: Why database management systems?

- Database management systems (DBMSs) are very good at organizing and managing large collections of persistent data.

We use DBMSs to help cope with large amounts of data because, when problems get big, they get hard.

- Consider the task of finding a particular book in a typical university library.
 - Now, reconsider that same task if the library doesn't keep the books arranged in any particular order or if the library has no indexes.
 - Using a big collection of unorganized things is practically impossible. Structure turns data into information.
 - **Persistence** means that the data exist permanently; they do not disappear when the computer is shut off.
- DBMSs are like suitcases: they are somewhere to put stuff so that it's all in one place and easy to get to.
 - DBMSs help protect data from unauthorized access.
 - DBMSs help protect data from accidental corruption or loss due to:
 - hardware failures such as power outages and computer crashes
 - software failures such as operating system crashes
 - DBMSs allow concurrent access, meaning that a single data set can be accessed by more than one user at a time
 - virtually all commercial database applications require the data entry staff to have access to the database simultaneously.
 - For example, an airline reservation system cannot restrict access to the database to a single travel agent.
 - concurrent data access introduces unwanted problems caused by two users manipulating exactly the same data at exactly the same time.
 - These problems can cause the database to be corrupted or for a user's interface program to never complete its query.
 - These problems are analogous to road intersections: if there are no traffic lights or stop signs, havoc will ensue.
 - DBMSs provide mechanisms to prevent concurrent access problems; these mechanisms are collectively called **concurrency control**.
 - A **distributed** DBMS allows a single database to be split apart such that its pieces reside at geographically separated sites.
 - this can provide performance improvements by eliminating transmitting the data across a relatively slow long distance communication channel (it's a lot faster to have the database on your hard drive than to access it across an Ethernet or via a modem)
 - this can reduce concurrency control bottlenecks by giving each user that part of the database which they need rather than having all the users compete for access to the whole database
 - DBMSs are not necessarily meant for data analysis; that is more the job of a spread sheet or some other special-purpose analysis tool.
 - DBMSs are general-purpose tools. It is basically irrelevant to the DBMS what is stored within it. Software design principles suggest de-coupling domain specific analysis packages from the DBMS to keep the division of labor clear.
 - DBMSs are very good at retrieving a relatively small portion of the database and passing it along for detailed analysis by a tool designed for that purpose.

- DBMSs often allow integrity constraints to be imposed on the data to insure validity and consistency. These rules can interfere with ad-hoc analysis in which the user manipulates the data without any preconceived ideas of how the data should relate to each other.
 - DBMSs often do not have adequate facilities to perform complicated calculations; some have no such facilities whatsoever.
-

2. Fundamental Concepts and Terminology

- This section presents a few common database concepts and terms.

2.1. Data

- Data are facts. Some facts are more important to us than others. Some facts are important enough to warrant keeping track of them in a formal, organized way.
- Important data are like the valuables we keep in a bank. They are a small subset of our total possessions but they are so important that we protect them by putting them in a special, safe place.
- “Data” is a plural. The singular of “data” is “datum”.
- "Data" is a broad concept that can include things such as pictures (binary images), programs, and rules. Informally, *data* are the things you want to store in a *database*.

2.2. Spatial vs. Non-spatial Data

- Spatial data includes location, shape, size, and orientation.
 - For example, consider a particular square:
 - its center (the intersection of its diagonals) specifies its location
 - its shape is a square
 - the length of one of its sides specifies its size
 - the angle its diagonals make with, say, the *x*-axis specifies its orientation.
- Spatial data includes spatial relationships. For example, the arrangement of ten bowling pins is spatial data.
- Non-spatial data (also called *attribute* or *characteristic* data) is that information which is independent of all geometric considerations.
 - For example, a person’s height, mass, and age are non-spatial data because they are independent of the person’s location.
 - It’s interesting to note that, while mass is non-spatial data, weight is spatial data in the sense that something’s weight is very much dependent on its location!
- It is possible to ignore the distinction between spatial and non-spatial data. However, there are fundamental differences between them:
 - spatial data are generally multi-dimensional and autocorrelated.
 - non-spatial data are generally one-dimensional and independent.

- These distinctions put spatial and non-spatial data into different philosophical camps with far-reaching implications for conceptual, processing, and storage issues.
 - For example, sorting is perhaps the most common and important non-spatial data processing function that is performed.
 - It is not obvious how to even sort locational data such that all points end up “nearby” their nearest neighbors.
- These distinctions justify a separate consideration of spatial and non-spatial data models. This unit limits its attention to the latter unless otherwise specified.

2.3. Database

- A database is a collection of facts, a set of data.
 - It is like the contents of a bank's vault.
- The information in a phone book is an example of a database.
 - Pay carefully attention to the fact that the book itself is not the database.
 - Rather, the database is the information stored on the pages of the book, not the pieces of paper with ink on them.

2.4. Repository

- A repository is a structure that stores and protects data.
- Repositories provide the following functionality:
 - add (insert) data to the repository
 - retrieve (find, select) data in the repository
 - delete data from the repository
- Some repositories allow data to be changed, to be updated.
 - This is not strictly necessary because an update can be accomplished by retrieving a copy of the datum from the repository, updating the copy, deleting the old datum from the repository, and inserting the updated datum into storage.
- Repositories are like a bank vault. They exist mainly to protect their contents from theft and accidental destruction.
 - Security: repositories are typically password protected, many have much more elaborate security mechanisms.
 - Robustness: Accidental data loss is safeguarded against via the ***transaction*** mechanism.
 - A ***transaction*** is a sequence of database manipulation operations.
 - Transactions have the property that, if they are interrupted before they complete, the database will be restored to a self-consistent state, usually the one before the transaction began.
 - If the transaction completes, the database will be in a self-consistent state.
 - Transactions protect the data from power failures, system crashes, and concurrent user interference.
- An example of a commercially available repository is Kala (Simmel and Godard 1991).

2.5. Database Management System (DBMS)

- A *database management system* is a data repository along with a user interface providing for the manipulation and administration of a database. A phone book is an example of a DBMS.
- Unless specified otherwise, a DBMS will hereafter be understood to be a software system, a program (or suite of programs) that is run on a digital computer. A few examples of commercially available DBMSs include Gemstone, O₂, Versant, Mattise, Codasyl, Sybase, Oracle, DB2, Access, and dBase.
- A DBMS is like a full-service bank, providing many features and services missing from the comparatively Spartan repository.

2.6. Queries

- Many DBMSs provide a user interface consisting of some sort of formal language.
- A *data definition language* (DDL) is used to specify which data will be stored in the database and how they are related.
- A *data manipulation language* (DML) is used to add, retrieve, update, and delete data in the DBMS.
- A *query* is often taken as a statement or group of statements in either a DDL or a DML or both. Some researchers view queries as read-only operations, no data modifications are allowed (Codd 1990, p. 21).
- A *query language* is a formal language that implements a DDL, a DML, or both. Examples of query languages include SQL (Structured Query Language), QUEL, ISBL, and Query-by-Example.

2.7. Data Models

- A *data model* is mathematical formalism consisting of two parts (Ullman 1988, p.32):
 - A notation for describing data, and
 - A set of operations used to manipulate that data.
- A data model is a way of organizing a collection of facts pertaining to a system under investigation.
- Data models provide a way of thinking about the world, a way of organizing the phenomena that interest us.
- They can be thought of as an abstract language, a collection of words along with a grammar by which we describe our subject.
 - By choosing a language, we pay the price of being constrained to form expressions whose words are limited to those in the language and whose sentence structure is governed by the language's grammar.
 - We are not free to use random collections of symbols for words nor can we put the words together in any *ad hoc* fashion.
- A major benefit we receive by following a data model stems from the theoretical foundation of the model.

- From the theory emerges the power of analysis, the ability to extract inferences and to create deductions that emerge from the raw data.
 - Different models provide different conceptualizations of the world; they have different outlooks and different perspectives.
 - There is no universally agreed upon best data model so this unit presents the most common ones.
 - DBMSs are seen to be composed of three *levels of abstraction*:
 - *physical*: this is the implementation of the database in a digital computer. It is concerned with things like storage structures and access method data structures.
 - *conceptual*: this is the expression of the database designer's model of the real world in the language of the data model.
 - *view*: different user groups can be given access to different portions of the database. A user groups portion of the database is called their *view*.
 - This unit is concerned mostly with the conceptual level.
-

3. Common Data Models

- This section presents an overview of the most common data models

3.1. Entity-Relationship Model

- The Entity-Relationship (ER) model is generally attributed to (Chen 1976).
- The ER model envisions the world as comprised of *entities* that are associated with each other by *relationships*. All of the entities of a particular type are collected together into *entity sets*.
- Entity sets and relationships can be depicted graphically in an [ER-diagram](#).

3.1.1. Entities

- Entities are distinguishable “real-world” objects such as employees, maps, airplanes, or bus schedules.
 - “Distinguishable” means that all entities can be uniquely identified.
 - Entities have common attributes that define what it means to be such an entity.
 - Any particular real-world object does not necessarily have a single or best representation as an entity.
 - For any given real-world object, different modelers can choose different sets of attributes of the object that are of interest to their particular situation.
 - This results in the same object being modeled differently.
- Entities are collected into *entity sets*.
 - Entity sets are depicted as rectangles in *ER diagrams*.
 - Their attributes are depicted as ellipses attached to the rectangles by lines.

3.1.2. Relationships

- A relationship is a list of entity sets.
 - Notation: two entity sets A and B that stand in relationship r is written $A r B$. See the next bullet for examples.
- Types of relationships (see [Figure 1.](#)):
 - aggregating relationships:
 - one-one: if $A r B$ and r is one-one then each entity of B is in relationship with at most one entity of A and vice-versa.
 - For example, if *CAPTAIN commands VESSEL* and **commands** is one-one then, in our model, each vessel has at most one captain and each captain commands at most one vessel at a time.
 - many-one: if $A r B$ and r is many-one then each entity of A is in relationship with at most one entity of B but not vice-versa.
 - For example, if *CREW assigned-to VESSEL* and **assigned-to** is many-one then, in our model, a vessel has many crew members but a crew member is assigned to only one vessel.
 - many-many: if $A r B$ and r is many-many then each entity of A can be in relationship with any number of B entities and vice-versa.
 - For example, if *VESSEL patrols REGION* and **patrols** is many-many then, in our model, a vessel patrols many regions and a region is patrolled by many ships.
 - **isa** (read “is a”) relationships: if $A \text{ isa } B$ then A is a specialization of B , or, conversely, B is a generalization of A .
 - For example, if *CAPTAIN isa CREW* then, in our model, captains have all the attributes of crew members but not vice versa.
 - The **isa** relationship allows hierarchies to be established among entity sets.
- A Relationship is depicted by a lozenge with lines connecting it to the relevant entity sets.
- The Entity-Relationship model lacks an underlying formalism and is, therefore, used more for general conceptualization than for creating physical models
 - (indeed, some authors do not acknowledge the ER model as a data model at all).
 - It is not uncommon for a conceptual design to be expressed in the ER model and then “translated” into another model for implementation.

3.2. The Network Model

- The network data model is based upon the concept of a *structure* such as is found in programming languages like C or Pascal.
 - ER entities can be modeled as structures with the entity’s attributes corresponding to the structure’s fields.
 - Entities are distinguished by their location, *i.e.*, the “physical” address of the

structure that is holding them. Thus, two structures of identical value represent two separate entities.

- Entity sets can be implemented as files whose records match the structures.
- Relationships are created with explicit linkages (*viz.* pointers) from structure to structure.
- Codasyl is an example of a DBMS based on the network model (Olle 1978).
- The network model has no formal semantics nor a high-level query language. Database manipulation was done via custom programs often written in COBOL.
- Network model databases are hand-coded and, therefore, can be very efficient in their space utilization and query execution times; all the relationships are “hardwired” or precomputed and built into the structure of the database itself.
- The price for such performance is inflexibility and great difficulty of use (among many other things).

3.3. The Relational Model

- The relational model was introduced by Codd (1970) and has been the inspiration of an entire generation of database management systems that are based on the concept of a *relation* which is a set of *tuples*.

3.3.1. Tuples

- A tuple is a set of facts that are related to each other in some way (perhaps only by the fact they’ve been put together in a set).
- Each fact in a tuple is a datum whose value comes from a specified domain (*e.g.*, the domain of all integers, the domain of all character strings of length 255 or less, *etc.*)
- Formally, let D_1, \dots, D_n be n sets of values constituting n domains (n is usually greater than zero but that is not strictly necessary). A *tuple* t is a set of values $t = \{d_1, \dots, d_n\}$, such that d_1 is an element of D_1, \dots , and d_n is an element of D_n . The domains are called *attributes*.

3.3.2. Relations

- Formally, let D_1, \dots, D_n be n domains. A *relation* R is a set of tuples over the Cartesian product $D_1 \times \dots \times D_n$.
- In English, a relation is a (possibly complete) subset of all the possible tuples formed by the Cartesian product of the domains.
- Since tuples are sets (of values) and a relation is also a set (of tuples), relations are sets of sets.
 - a file is a list of records

- a table is a list of rows
- a relation is a set of tuples
- Relations are naturally represented as tables.
 - Tables are not relations because relations cannot have duplicate tuples and there is no such stricture on tables. However, it is perhaps convenient to think about relations as tables so long as the distinction remains clear.
 - Most (if not all) commercial “relational” DBMSs violate this principle: they allow duplicate tuples.
- The use of relations as a data modeling tool becomes apparent when we have a relation, say, “*OUR_DEM*” with fields {*quadname*, *zone_code*, *mappingcenter*}.
 - It happens that the USGS has a digital elevation model named “PLACITAS NM” in UTM zone 13 that was created by the Forest Service Mapping Center.
 - Then, the presence of a tuple in the *OUR_DEM* relation whose
 - *quadname* attribute has the value “PLACITAS NM” and
 - *zone_code* attribute has the value “13”, and
 - *mappingcenter* attribute has the value “FS”,
 - indicates that we have the Placitas DEM in our possession.

3.3.3. Tuples, Relations and Keys

- Relations are *sets* of tuples; consequently, no two tuples that are elements of the same relation can have identical values for all their attributes. That is to say, there are no duplicate tuples in a relation.
- All tuples in a relation can be distinguished by the values of their attributes.
 - Any set of attributes whose values *necessarily* uniquely identify a tuple are said to be a *key*.
- Database designers choose some attribute set to be a key for their database’s relations.
 - This key is known as the ***primary key***.
- If the primary key of one table appears as an attribute of a different relation, the key is known as a ***foreign key*** in the other relation.
- A key uniquely identifies its tuple. Therefore, a tuple’s key is often used as a surrogate for the entire tuple.

3.3.4. Relationships

- Not surprisingly, the relational model represents relationships with relations.
- [Figure 2](#) depicts a relational database that was designed from the ER-diagram developed above.
 - Key attributes are denoted in bold face.

- Aggregating relationships are represented by embedding the primary key of one relation into another relation as a foreign key:
 - *one-one*: if $A \mathbf{r} B$ and \mathbf{r} is one-one then the primary key of A can be embedded in B or vice versa or both.
 - For example, suppose **CAPTAIN** **commands** **VESSEL** and that **commands** is one-one.
 - Suppose further that *cptn_name* is the primary key of **CAPTAIN** and *vessel_name* is the primary key of **VESSEL**.
 - Then, **CAPTAIN** could have an attribute *commands* whose value is that of *vessel_name* for the vessel that captain commands.
 - It is equally reasonable to have an attribute *commanded_by* in **VESSEL** whose value is that of name for the captain commanding the vessel.
 - *many-one*: if $A \mathbf{r} B$ and \mathbf{r} is many-one then the primary key of B can be embedded in A but not vice versa.
 - For example, suppose **CREW** **assigned-to** **VESSEL** and **assigned-to** is many-one.
 - Suppose further that *crew_name* is the primary key of **CREW** and *vessel_name* is the primary key of **VESSEL**.
 - Then, **CREW** could have an attribute *assigned_to* whose value is that of *vessel_name* for the vessel this crew member serves on.
 - However, **VESSEL** cannot have an attribute *roster* because *roster* would have to be a set (many crew members per vessel) and the relational model stipulates that all domains are atomic; no collections.
 - *many-many*: if $A \mathbf{r} B$ and \mathbf{r} is many-many then neither primary key can be embedded the other table. Again, the difficult lies in the atomicity rule for domains. So, for a many-many relationship, we must create a separate relation whose attributes include but are not limited to the primary keys from A and B .
 - For example, if **VESSEL** **patrols** **REGION** and **patrols** is many-many.
 - Suppose further that *vessel_name* is the primary key of **VESSEL** and *region_name* is the primary key of **REGION**. Then we have a third relation **PATROLS** with attributes *vessel_name* and *region_name*.
 - such relations are sometimes called **join supports**
 - such relations are no different in any way from any other relation
- **isa** relationships are handled as the other relationships:
 - one-one: Suppose **CAPTAIN** **isa** **CREW**.
 - Then there is a one-one relationship between **CAPTAIN** and **CREW** so the primary key of **CREW** can be used as the key in **CAPTAIN**.
 - The one-one nature of this relationship indicates that the two tuples really give details of the same entity; they are sort of like a single tuple that has

been split in two.

- many-one: Suppose we are modeling WWII combat vessels, known collectively as "ship(s) of the line" (*SOTL*). It happens that a ship design can be used as the plan for many individual vessels (obviously).
 - The design is known as a “class” and the vessels made to that design are said to belong to that class.
 - For example, the USS Missouri belongs to the Iowa class of battleships.
 - We model this relationship with a relation *SOTL* which has a single tuple for each class of warship. Thus, *VESSEL isa SOTL*.
 - The *SOTL* relation has attributes that are common to all ships of the line. For WWII vessels, this might include attributes such as the number of primary guns, size of the primary guns, *etc.*
 - The tuple in *SOTL* for the Iowa battleships gives information that is common to all Iowa class battleships (*e.g.*, nine 16-inch guns, *etc.*).
 - The tuple in *VESSEL* for the USS Missouri holds the information specific to that vessel including the fact it belongs in the Iowa class.
 - Therefore, the primary key of *SOTL* is embedded in *VESSEL*, not vice versa.
 - Compare many-one **isa** relationships with one-one **isa** relationships.
 - Ullman restricts relationships to be one-one (Ullman 1988, p. 35).

3.3.5. Query Languages

- Codd invented two early languages for dealing with relations: one was algebraic and the other was based on first-order predicate logic (Codd 1971). These languages have the same expressive power.
 - *Relational Algebra*
 - “[an] algebraic notation ... where queries are expressed by applying specialized operators to relations” (Ullman 1988, p. 53)
 - see (Codd 1990, pp. 61-144) for a presentation of the relational algebra.
 - *Relational Calculus*
 - “[a] logical notation ... where queries are expressed by writing logical formulas that the tuples in the answer must satisfy” (Ullman 1988, p. 53)
 - see (Ullman 1988, pp. 145-160) for a presentation of the relational calculus.
- The most common commercial query language is the *Structured Query Language*, or *SQL*.
 - Despite its reputation as a relational query language, SQL does not fully support the relational model (it includes things that are not in the model and omits things that are. See (Codd 1988) and (Date 1987)).

3.3.6. Relational Database Management System (RDBMS)

- A *relational database management system* is a DBMS based on the relational model as

defined by (Codd 1990).

- There is no commercially available DBMS that fully implements the relational model as defined by (Codd 1990). Some are coming closer. Not everyone agrees that this strict lack of conformance is a Bad Thing.

3.3.7. Advantages of the Relational Model

- Codd (1990, pp. 431-439) presents many advantages of the relational model. Some of them are highlighted below:
 - The relational model is truly a mathematically complete data model. This solid theoretical underpinning is responsible for
 - *ad hoc* query languages whose queries can be automatically compiled, executed, and optimized without resorting to programming
 - correctness: the semantics of the relational algebra are sound and complete
 - predictable: the consistent semantics enables users to easily anticipate the result of a given query
 - Adaptability: making a change in the structure of the tables in the network model requires programmatic making changes to all the database's queries. As a result, the network model is inflexible in the extreme.
 - The relational model cleanly separates the logical from the physical model and this decoupling mitigates or eliminates these problems.
 - Also, the relational model's integrity constraints are very helpful in ensuring that structural changes did not adversely effect the meaning of the database.
 - Multiple views: it is straightforward to present different user groups different views of the same database.
 - Concurrency: a full theory of transaction concurrency control exists which depends upon the theoretical formalisms of the relational model.
 - This theory guarantees the correct execution of concurrent queries (indeed, it defines what "correct" means!)

3.4. The Object Model

3.4.1. What is the Object Model?

- The word "object" is similar to the Entity-Relationship concept of an "entity" although "object" is more general.
 - I recommend taking "object" in the spirit of "objects in the physical world."
 - Objects are things but they are not limited to physical, tangible things. For example, data structures (*e.g.*, a hash table) can be objects.
 - All objects are distinct and, like the network model, are made distinct by an identifying attribute, the *object ID*.

- Like the other models, the object model assumes that objects can *conceptually* be collected together into meaningful groups. These groups are called *classes*.
- An object grouping is meaningful because objects of the same class must have common attributes, behaviors, and relationships with other objects.

- Unlike entity sets and relations, classes do not actually hold the objects of that class.
 - Classes are purely conceptual.
 - There is nothing in the object model that is equivalent to either a entity set or a relation (there could be but it's not required by the model).
- Like the network model, the relationships among objects are specified via a “physical” link (pointer) between objects.
- According to Rumbaugh *et al.* (1991), “The object model describes the structure of objects in a system – their identity, their relationships to other objects, their attributes, and their operations.”

- The DARPA Open OODB project proposes the following as the essential features of the OO data model (Blakeley 1991) and (Rao 1994, p.72):
 - *Object identity*: the ability of the system to distinguish between two different objects that have the same state. The state of an object can be shared by several objects via object identity.
 - *Encapsulation*: a kind of abstraction that enforces a clean separation between the external interface (behavior) of an object and its internal implementation. Encapsulation requires that all access (or interaction) with objects be done by invoking the services provided by their external interface.
 - *Complex state*: the ability to define data types whose implementation has a nested structure. The state of an object could be built from records of primitive types, other objects, or [collections] of objects.
 - *Type extensibility*: the ability to define new data types from previously defined types by enhancing or changing the structure or behavior of the types. Type inheritance is a mechanism used to define new types by enhancing already existing behavior.
 - *Genericity*: The types of the object data model with which the object query language collaborates must be generic. That is, as a new type is added to the system, it must be querable.

- There is no universally agreed upon object data model but *The Object-Oriented Database System Manifesto* (Atkinson, *et al.* 1989) gives a framework being considered from which to derive a standard.

- According to Rao (1994), “The object-oriented database (OODB) paradigm is the combination of object-oriented programming language (OOPL) systems and persistent systems. The power of the OODB comes from the seamless treatment of both persistent data, as found in databases, and transient data, as found in executing programs.”

- Note that the emphasis with OODB, like the network model, is towards programmers, not end users.
 - This point is further emphasized by the primary interface to OODBs being OOPLs.
- I suggest (Booch 1994) for a good introduction to object-oriented design and analysis.

3.4.2. Inheritance (isa) Relationships and Typing

- Many object-oriented models take classes to be a typing mechanism (for example, Eiffel (Meyer 1997) and C++ (Stroustrup 1997)).
- The *type* of an object is its *class*; an object is an *instance* of its class.
 - For example, the number 2.3 is an instance of the class of rational numbers.
- Interpreting classes to be types implies the inherent ability of users to create their own data domains.
- Inheritance can be viewed from two perspectives (Cusack 1991):
 - **incremental**: the process of adding attributes and functions to an existing class (the **base class**).
 - new attributes/functions can added to the new class that were not in the base class.
 - this is a technique for code reuse.
 - no typing information is implied by this relationship.
 - for example, suppose that there is a class **PERSISTENT** that has the functionality of automatically storing its objects in a database. Any class that inherits **PERSISTENT** “magically” gains the ability to do likewise.
 - **subtyping**: a technique for arranging class definitions in a hierarchy satisfying the condition that members of the subclass are also members of the superclass.
 - subtyping constitutes the **isa** relationship.
 - old attributes/functions can change type so long as the new type is more specific (it inherited either directly or indirectly) than the original base class.
 - old attributes/functions cannot be removed.
 - old functions can be provided with new implementations so long as the interface to the function remains unchanged (or is changed via specialization as indicated above)
- Various object models span the gambit of inheritance relationships:
 - full repeated multiple inheritance (Eiffel, C++)
 - single inheritance (Java)
 - no inheritance (Actor, Ada)

3.4.3. Encapsulation

- Objects *encapsulate* their attributes and the behaviors. This implies:

- there is no interaction with an object that does not go through a publicly published interface
- objects manipulate their own state; the definition of class includes the object's behavior manifested as functions and procedures.
- an object's state *cannot* be manipulated by anything external to them (at least, not without permission).
- For example, in a non object-oriented language such as C, let's say a programmer writes a procedure to change the values of a structure holding the position of a graphics primitive.
 - In an object-oriented language, the programmer creates a graphics primitive class that has its positional information along with an internal procedure that changes its own position.
 - The programmer “sends a message” to the object requesting it to change its own position.
- The advantage of encapsulation is that the implementation of any behavior can be changed without effecting any other class in the system. This helps de-couple the classes and reduces the complexity of the system.

3.4.4. Comparison to the Relational Model

- The object model differs from the relational model in (at least) the following ways:
 - The object model allows complex objects to be attribute domains; this is prohibited in the relational model.
 - The only complex type available in the relational model is the relation.
 - The object model restricts all system entities to be objects which is a more general concept than a relation (relations can be objects but not all objects are relations).
 - The relational model allows no duplicate tuples and, consequently, entities are identified by their attribute values.
 - The object model assumes the existence of an object ID which uniquely identifies its object and is, possibly, invisible to the user.
 - Objects are instances of classes and classes constitute the typing system of the model.
 - There is no concept of class-level typing in the relational model; everything is a relation.
 - The relational model supports user-defined domains but this is applied at the attribute level whereas, with the object model, the class is also a type.
 - The equivalent in the relational world would be for relations to constitute types, as well.
 - There is no generally accepted formal object model.
 - The relational model is well-defined, sound, and complete.
 - Relations hold all tuples. There is no equivalent for objects; there is no set or anything else that contains all the objects of a class.
 - There are many higher-order, non-programming query languages for the relational model. There are few equivalents for the object model (UniSQL is an example).
 - The object model is aimed more at programmers than at end users; the reverse is true of the relational model.

4. Summary

- This unit has presented many of the more common reasons why people need and use database management systems.
 - A DBMS provides for the storage, retrieval, removal, and analysis of large quantities of data.
 - A DBMS provides safety and security from accidental loss or theft of data.
 - A DBMS is analogous to a full-service bank:
 - data are like valuables,
 - a database is the collection of all the valuables stored in the bank,
 - a repository is like a vault,
 - a DBMS is like the entire, full-service bank.

- The most common data models
 - Entity-Relationship (ER)
 - real-world things are modeled by *entities*.
 - all entities of the same type are collected together into an *entity set*.
 - the relationships between entity sets are represented by *relationships*.
 - Network
 - essentially a programmer's database model
 - efficient but inflexible and hard to understand
 - Relational
 - its only complex data type is the *relation*
 - it is the only complete data model
 - aimed at users instead of programmers
 - relational query languages are easier to use than full-blown programming languages
 - rich underlying theory
 - separation of implementation and design
 - Object-Oriented
 - an extension of object-oriented programming
 - no generally agreed upon formal data model
 - great freedom regarding complex data structures
 - inheritance
 - user-defined types
 - encapsulation

5. Review and Study Questions

5.1. Essay and Short Answer Questions

- What is the difference between a spread sheet program and a database management system? When would you use one or the other?
- Why bother with data modeling? Is there anything "wrong" with just putting data into

- the database in whatever way seems good at the moment?
- What are the strengths and weaknesses of the Entity-Relationship data model?
 - What are the strengths and weaknesses of the Network data model?
 - What are the strengths and weaknesses of the Relational data model?
 - What are the strengths and weaknesses of the Object-Oriented data model?
 - The relational model uses "value" identity, meaning that entities can be distinguished by examining the values of their attributes. Neither the network nor the object model follow this approach.
 - First: do you feel that this distinction is significant? Why or why not?
 - Second: What are the advantages and disadvantages of each approach?
 - Explain the distinction between aggregate and inheritance relationships.
 - What is a "complex" object?
 - It is always possible to find a primary key for *any* relation. Why?
 - Why are tables not relations?

5.2. Multiple-choice questions

- What relationship does a particular, individual book stand in with respect to libraries?
 1. one-one
 2. many-one
 3. one-many
 4. many-many
- What relationship does a book title stand in with respect to libraries?
 1. one-one
 2. many-one
 3. one-many
 4. many-many

Choose the best or most appropriate answer(s) to the question.

6. Bibliography

- Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., and Zdonik, S. (1989). The Object-Oriented Database System Manifesto. In *Proceedings DOOD '89*, Kyoto, December.
- Blakeley, J.A. (1991) DARPA Open Object-Oriented Database Preliminary Module Specification: Object Query Module. Texas Instruments, Inc., Version 3, Nov. 25.
- Booch, G. (1994) *Object-Oriented Analysis and Design with Applications, 2nd ed.* Benjamin/Cummings Publishing Company.
- Chen, P.P. (1976) The Entity-Relationship Model – Toward a Unified View of Data. *ACM TODS*, 1:1.
- Codd, E.F. (1970) A Relational Model of Data for Large Shared Data Banks. *Comm. ACM* 13:6.
- Codd, E.F. (1971) ALPHA: A Data Base Sublanguage Founded on the Relational Calculus. In *Proc. 1971 ACM SIGFIDET Workshop* (San Diego, Nov. pp. 11-12).

- Codd, E.F. (1988) Fatal Flaws in SQL (Both the IBM and the ANSI Versions). *Datamation*, August and September.
- Codd, E.F. (1990) *The Relational Model for Database Management, version 2*. Addison-Wesley Publishing Company, New York.
 - Cusack, E. (1991) Inheritance in Object-Oriented Z. In Pierre America (ed.), *Lecture Notes in Computer Science, ECOOP '91, European Conference on Object-Oriented Programming*, Springer-Verlag.
 - Date, C.J. (1987) Where SQL Falls Short. (abridged) *Datamation*, May 1. Unabridged version, What is Wrong with SQL, available from The Relational Institute, San Jose.
 - Meyer, B. (1997) *Object-Oriented Software Construction*, 2nd ed. Prentice Hall.
 - Olle, T.W. (1978) *The Codasyl Approach to Data Base Management*. John Wiley & Sons, New York.
 - Rao, B.R. (1994) *Object-Oriented Databases: Technology, Applications, and Products*. McGraw-Hill, Inc., New York.
 - Rumbaugh, J., Blaha M., Premerlani, W., Eddy, F., and Lorensen, W. (1991) *Object-Oriented Modeling and Design*. Prentice Hall, New Jersey.
 - Simmel, S.S. and Godard, I. (1991) The KALA BASKET: A Semantic Primitive Unifying Object Transactions, Access Control, Versions and Configurations. In *Proceedings of OOPSLA '91: Conference on Object-Oriented Programming Systems, Languages and Applications*, Nov., pp. 230-246.
 - Stroustrup, B. (1997) *The C++ Programming Language*. Addison-Wesley Publishing Company.
 - Ullman, J.D. (1988) *Principles of Database and Knowledge-Base Systems, volumes I and II*. Computer Science Press, Inc.
-

7. Reference Materials

7.1. Print References

Bancilhon, F., Delobel, C., and Kanellakis, P. (1992) *Building an Object-Oriented Database System: The Story of O2*. Morgan Kaufman Publishers. O2 is an important and powerful object-oriented database management system. This book provides many interesting insights into O2 in particular and into one approach to OODBMSs in general.

Booch, G. (1994) *Object-Oriented Analysis and Design with Applications, 2nd ed.* Benjamin/Cummings Publishing Company. There are many excellent titles covering object-oriented design and analysis; this is just one of them. I believe it provides an outstanding introduction to the topic for both seasoned programmers and object newcomers alike.

Chen, P. (1977) *The Entity-Relationship Approach to Logical Data Base Design*. QED Publishing Co. This is a good introductory book by the man who is generally attributed as having developing the E-R modeling approach. It is out of print and, therefore, can be hard to find.

- Codd, E.F. (1990) *The Relational Model for Database Management, version 2*. Addison-Wesley Publishing Co., New York. This work is not an introductory text and might not be suited for beginners. It is, however, a must for anyone who is seriously interested in the relational model either from an implementation or from a theoretical point of view.
- Date, C.J. (1994) *An Introduction to Database Systems*. Addison-Wesley Publishing Co., New York. A comprehensive and approachable classic that covers a broad spectrum of database issues. Date covers both relational and object-oriented databases with material that is suitable for experts and beginners alike. This book covers the essential core of the relational model with special attention to practical implementation issues, however, it is not specifically aimed at designing databases.
- Hernandez, M.J. (1997) *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design*. Addison-Wesley Publishing Co., New York. This book is aimed, for the most part, at beginning relational database designers. It presents a platform independent approach while avoiding lots of jargon and theory that is irrelevant in the early stages.
- Kim, W. (1990) *Introduction to Object-Oriented Databases*. MIT Press. Dr. Kim is a widely respected researcher and developer of object-oriented theory and systems. He is a pioneer in that part of database theory which is trying to bridge the gap between the object-oriented world and the relational world.
- Maier, D. (1983) *Theory of Relational Databases*. Computer Science Press. For those interested in the formal underpinnings of the relational database model, this is the book.
- Meyer, B. (1997) *Object-Oriented Software Construction, 2nd ed.* Prentice Hall. The Eiffel programming language is notable for its small size, expressive completeness, simplicity, elegance, and practicality. This book presents the Eiffel language as a vehicle with which to discuss software engineering.
- Stroustrup, B. (1997) *The C++ Programming Language, 3rd ed.* Addison-Wesley Publishing Company. The C++ programming language is almost ubiquitous and this book is the C++ Bible. Bjarne Stroustrup created C++ and this book represents the latest in a long line of texts he has written on the subject.
- Ullman, J.D. (1988) *Principles of Database and Knowledge-Base Systems, volumes I and II*. Computer Science Press, Inc. If I were constrained to own only one title about database systems, these two volumes would be it. They cover every aspect of the topic from the theory to the implementation along with the connections from databases to knowledge-bases and the artificial intelligence community.
-

Citation

To reference this material use the appropriate variation of the following format:

Meyer, Thomas H. (1997) Non-spatial Database Models, *NCGIA Core Curriculum in GIScience*, <http://www.ncgia.ucsb.edu/giscc/units/u045/u045.html>, posted November 10, 1997.

Last revised: November 19, 1997.

Non-Spatial Database Models (045)

Instructors' Notes

- This section introduces the terms and concepts needed to understand non-spatial databases and their underlying data models, including a motivation of the need for database management systems, an overview of database terminology, and a description of non-spatial data models. After learning the material covered in this unit, students should be able to explain the purpose of a database management system, list the major non-spatial data models and their features, and identify the primary distinctions between the major non-spatial data models.
-

Unit 045 - Non-spatial Database Models

Table of Contents

Advanced Organizer

- Topics covered in this unit
- Intended learning outcomes
- Instructors' notes
- Metadata and revision history

Body of unit

1. Motivation
2. Fundamental Concepts and Terminology
 1. Data
 2. Spatial vs. Non-spatial Data
 3. Database
 4. Repository
 5. DBMS
 6. Queries
 7. Data Model
3. Common Data Models
 1. Entity-Relationship Model
 1. Entities
 2. Relationships
 2. The Network Model
 3. The Relational Model
 1. Tuples
 2. Relations
 3. Tuples, Relations and Keys
 4. Relationships
 5. Query Languages
 6. RDBMS
 7. Advantages of the Relational Model
 4. The Object Model
 1. What is the Object Model?
 2. Inheritance Relationships and Typing
 3. Encapsulation
 4. Comparison to the Relational Model
4. Summary
5. Review and Study Questions
6. Bibliography

Citation

[Back to the Unit](#)

Unit 045 - Non-Spatial Database Models

Metadata and Revision History

1. About the main contributors

- author
 - Thomas H. Meyer, Mapping Sciences Laboratory

Texas A&M University, USA

2. Details about the file

- unit title
 - Non-Spatial Database Models
- unit key number
 - 045

3. Key words

4. Index words

5. Prerequisite units

6. Subsequent units

7. Other contributors to this unit

8. Revision history

- 19 November 1997 - revised draft
-

Figure 1: Entity-Relationship Diagram Example

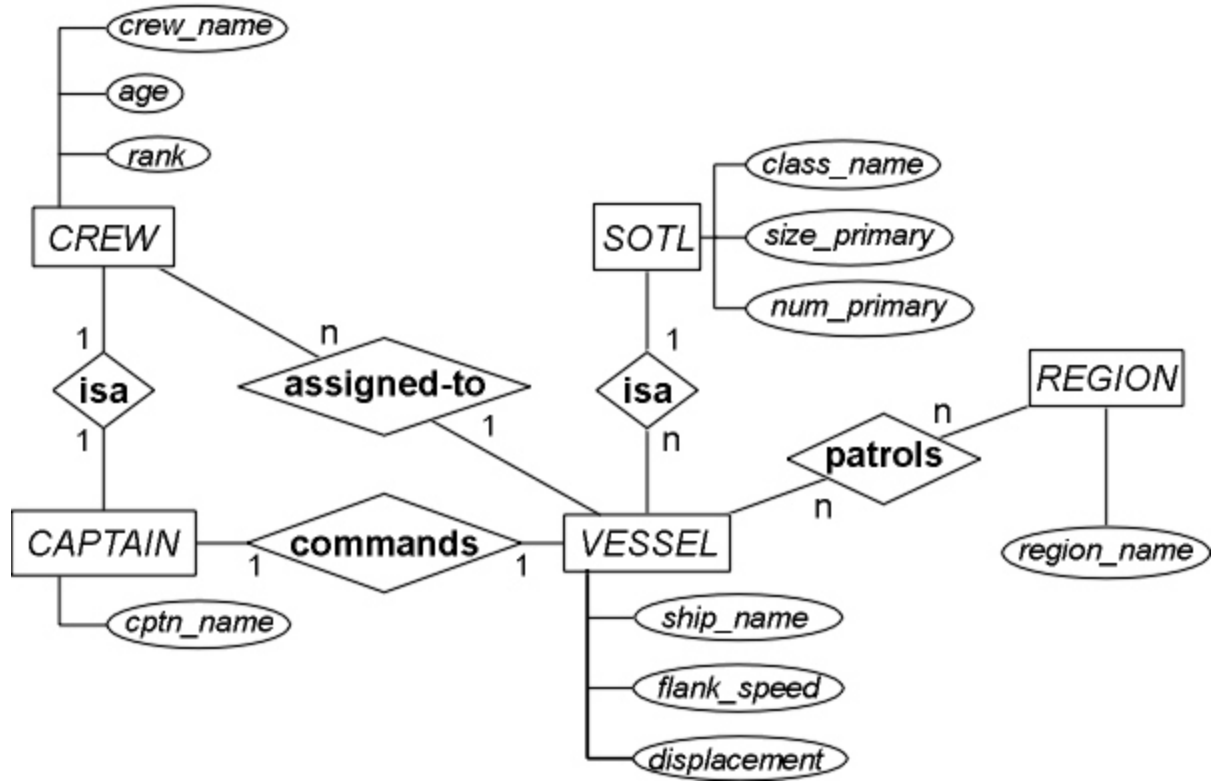


Figure 2. Example of relations and their relationships

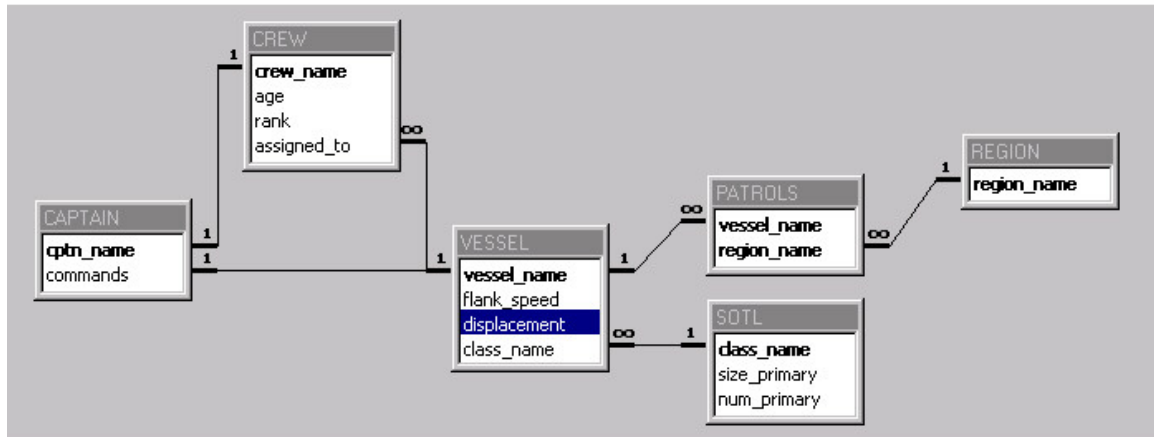


Figure 3: The CAPTAIN Relation

	cptn_name	commands
	Almquest E	Maryland
	Ericksson M	Washington
	Geary F	Idaho
	Harrison F	Mississippi
	Johnson M	New Mexico
	Knox B	Colorado
	Wheeler P	West Virginia

Record:

Figure 4: The VESSEL Relation

	vessel_name	flank_speed	displacement	class_name
	Colorado	21	32600	Colorado
	Idaho	21	32000	New Mexico
	Maryland	21	32600	Colorado
	Mississippi	21	32000	New Mexico
	New Mexico	21	32000	New Mexico
	Washington	21	32600	Colorado
	West Virginia	21	32600	Colorado
▶		0	0	

Record: 8 of 8

Figure 5: The CREW Relation

CREW : Table				
	crew_name	age	rank	assigned_to
	Almquest E	59	Captain	Maryland
	Cutler M	19	Cook	New Mexico
	Ericksson M	53	Captain	Washington
	Geary F	57	Captain	Idaho
	Harris N	18	Cook	Idaho
	Harrison F	49	Captain	Mississippi
	Jackson L	21	Bosun	Maryland
	Jackson R	22	Helmsman	Washington
	Johnson M	54	Captain	New Mexico
	Knox B	55	Captain	Colorado
	Wheeler P	61	Captain	West Virginia
▶		0		

Record: ⏪ ⏩ 12 ▶ ▶▶ ▶* of 12

Figure 6: The REGION Relation



region_name
Leyte Gulf
Guadalcanal Straits
San Diego
Pearl Harbor
Guam
Midway

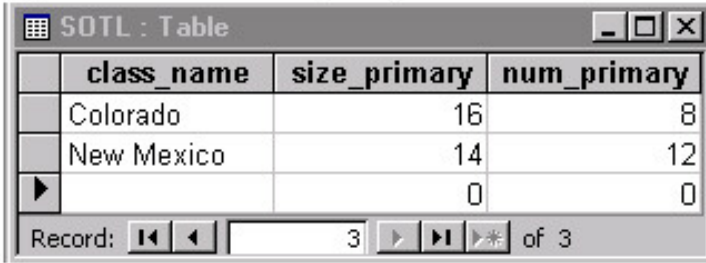
Record:

Figure 7: The PATROLS Relation



vessel_name	region_name
Colorado	Leyte Gulf
Idaho	Guadalcanal Straits
Maryland	San Diego
Mississippi	Leyte Gulf
New Mexico	Midway
Washington	Leyte Gulf

Record: 7 of

Figure 8: The SOTL (Ship of the Line) Relation



	class_name	size_primary	num_primary
	Colorado	16	8
	New Mexico	14	12
▶		0	0

Record:  3  of 3