Database Concepts
Discussed in an Object Oriented Perspective

Yngve Lindsjørn
Norwegian Computing Center
P.O. Box 114 Blindern
N-0314 OSLO 3, Norway

Dag Sjøberg
The Central Bureau of Statistics
P.O. Box 8131 Dep.
N-0033 OSLO 1, Norway

Abstract

The terminology in the area of databases and data models is inconsistent and inaccurate and thus often confusing. Some fundamental database concepts are described in this paper. The description of these concepts are based on general concepts related to the development of the object oriented languages SIMULA, DELTA and BETA. A database is defined as an information system providing information about a referent system (the modelled part of the world), and a data model is defined as "having an inherent structure and a set of tools and techniques used in the process of designing, constructing and manipulating model systems (in particular, databases)". The connection between databases, DBMS and data models and other concepts related to the development and use of databases are described in a process/structure hierarchy.

1. Introduction

The term "object oriented" is derived from the SIMULA67 programming language [*Dahl, Myhrhaug, Nygaard, 68*]. Later, other object oriented programming languages appeared. In particular, Smalltalk [*Goldberg, Robson, 83*], normally associated with an (object oriented) graphical environment, has contributed to the popularity of the object oriented paradigm.

As will be illustrated in this paper, concepts associated with the object oriented paradigm are also suitable for describing databases. The paper aims at describing a *conceptual basis* for object oriented databases and environments where ideas from the object oriented programming style are integrated with facilities provided by modern database management systems (DBMS).

Most literature concerning object oriented databases follow the Smalltalk tradition (e.g. [*Copeland, Maier, 84*], [*Maier, Stein, 87*]). Some literature do not follow any specific tradition, but are influenced by several object oriented languages (e.g. [*Zdonik, Wegner, 86*], [*Skarra, Zdonik, 87*]). This paper follows the tradition of SIMULA, DELTA and BETA [*Dahl, Nygaard, 65*], [*Dahl, Myhrhaug, Nygaard, 68*], [*Delta,*

75], *[BETA, 83a]*, *[BETA, 83b]*, *[BETA, 85]*, *[BETA, 87]*, and the development of the basic concepts presented in Section 2 is related to the development of these languages.

Following this tradition, the *object oriented perspective* can briefly be interpreted as follows:

Information processes may be understood, organized, constructed and operated in terms of a system concept in the sense that the system is "the part of the world" that develops through the process. Program executions and operations of a database are examples of such processes. The substance of the process is organized as the components of the system, called objects. Any measurable property of the substance is a property of an object. Any transformation of state is regarded as action by objects.

Some of the concepts in this paper are more extensively discussed in *[Lindsjørn, Sjøberg, 87]*.

2. Basic Concepts

Using concepts from the object oriented perspective, we present definitions of basic concepts which constitute a basis for specialized database concepts. Most of the concepts defined here are presented earlier, see e.g. *[Nygaard, 86a]*, *[Nygaard, 86b]*, *[Delta, 75]*.

The first concept defined is *process*:

A **process** is a development of a part of the world through *state transitions* during a time interval.

An example of a process is the economic development of Norway after the Second World War. The concept process comprises its sequences of changing states.

A process is always restricted in its behaviour, and the *structure of a process* is defined as:

The **Structure** of a process is the limitation of its set of possible states and state transitions.

When regarding the economic development of Norway, the general economic policy of the Norwegian government, the laws, the total manpower, etc. are *structuring* the process.

In informatics, the processes dealt with are *information processes*:

A process is regarded as an **information process** when the qualities considered are:
- its *substance*, the physical matter that it transforms,
- *measurable properties* of its substance, represented by values,
- *transformations* of its substance and thus its measurable properties.

An essential concept in this paper is the *system* concept with its close relation to the object oriented perspective.

A **system** is a part of the world that a person (or group of persons, during some time interval and for some reason) chooses to regard as a whole, consisting of
- *components*, each component characterized by
- *properties* that are selected as being relevant and by
- *state transitions* relating to these properties and to other components and their properties.

According to this definition, no part of the world is a system as an inherent property. It is a system if we choose a system perspective.

In informatics, the systems dealt with are information systems:

> An **information system** is a system where the components are organized as objects, and the properties of the objects are *attributes* that are either *quantities*, *references* or *categories*.
>
> A **state** of an information system is expressed by describing
> - the *moment* at which the state is recorded,
> - its *objects*, the *states of attributes*, and *transitions* going on, all at that moment.

The transitions are actions that result in changes of state of the information system. The update of objects in a system (e.g. accounts in a bank system) is an example of a transition. The assignment x+1 -> x is another.

Properties of components are captured by attributes:

> **An attribute** of some object is the association of the object and:
> - a name,
> - a set of elements (the domain),
> - (at any given time) one of the elements of the set.

The attributes (of the objects) are divided into three kinds - quantities, references and categories.

> A **quantity** is an attribute for which:
> - a way of *measuring* a state of the attribute is defined,
> - there exists a *mapping* from the set of measurements of every possible state to the elements of the attribute's associated set, called the *value set*,
> - the associated element, called the attribute's *value*, corresponds to the measurement of the property at the given moment.

A *quantity* is used to denote values of measurements and is either a *variable* or a *function*.

> A **reference** is an attribute for which the associated set has objects as elements. The associated element is called the attribute's *referent*.

As it appears from the definition, the referent of a reference is the object referred to at a given time. The referent could be denoted (indicated) by a name or an address. However, we emphasize that the attribute does not have a "value" being the name or the address, but denotes the actual (referred) object - the referent. Just as a value can be assigned to a quantity-variable, an object can be assigned to a reference-variable.

> A **category**[1] is an attribute for which the associated set consists of prescriptions for *common structure* of categories of objects.

Procedures and functions are examples of category-attributes.

---

1 In BETA the concept *pattern* is used instead of *category*.

Analogous to the definition of structure of a process, we define the *structure of an information system*:

> The **Structure** of an information system is the limitation of its set of possible states and state transitions.

The structure is given by the category descriptions of the objects and their attributes with associated sets.

2.1 Model System - Referent System

We understand the concept "model" as an analogue device - a model is *similar* to "something". We find another use of the term in the concept "data model". A data model specifies rules for modelling and manipulating objects in an information system.

We now define the concepts *model* and *referent*:

> A phenomenon M is a **model** of a phenomenon R, according to some perspective P, if a person regards M and R as similar according to P. We call R the *referent phenomenon* (or simply *referent*) and M the *model phenomenon* (or simply *model*). We call P the perspective of the model M.

As mentioned, our object oriented perspective implies that we focus upon the *system* concept, and the phenomena we discuss in this paper are regarded as (information-) systems. When we choose to regard "a part of the world" as a system, we always produce another, structurally similar system - mental or manifest - as a vehicle for relating to that "part of the world". The "part of the world" system will be called the *referent system*, the mental or manifest system produced, will be called the *model system.*

An example:
A database, containing information about employees, departments, offices and items (and their selected attributes and constraints), is a *model system* of some part of an enterprise. The selected part of the enterprise is the *referent system* and the *database* constitutes the (manifest) model system.

2.2 Categories and Abstraction Mechanisms

One way of structuring and presenting the information contents of objects, their attributes and actions is through the use of abstraction. Abstraction provides the ability to hide details and concentrate on describing the common structure of similar phenomena. We briefly present the three commonly known abstraction mechanisms - *classification, generalization* and *aggregation.*

The category construct unifies the constructs (abstractions) class, type and procedure by means of a category description which prescribes common structure of a category of objects[2]. These objects are said to be *category specified* (as opposed to *singular objects* which are specified together with their descriptors - as for instance inner block instances in ALGOL).

2 In this context, quantities and actions (procedure/ function activations) are regarded as objects *per se.*

Below we present three different kinds of classification:

1. A *class* is intended to abstract *substance* (the substance is focused). Substance is organized as objects, and when we talk about objects, we normally mean instances of classes.
2. A *type* is intended to abstract *measurable properties* (quantities) of substance (values are focused).
3. A *procedure* is intended to abstract *actions* (actions are focused).

We refer to these categories as *class-category* (we also use *object-category*), *type-category* and *procedure-category*, respectively. We will also introduce a fourth kind of category - the *relationship-category*.

Categories can be organized in a hierarchy. This is referred to as *generalization/specialization*. A category can be a generalization of two or more categories. Conversely, categories are specialized from other categories (this can be done successively). If category B is a specialization of category A, B *inherits* all attributes (and actions) of A.

Most object oriented programming languages support generalization/specialization for class-categories only. As in BETA, this abstraction mechanism should be supported in all three kinds of categories (class-category, type-category and procedure-category).

Aggregation is the abstraction mechanism by which a category can be constructed (aggregated) from other categories (a category can also be regarded as aggregation of its constituent attributes). In BETA (and SIMULA) aggregation can be done successively. In most data models aggregation can be done at one level only.

In practise, there is no clear distinction between the process of aggregation and the process of classification. Normally, aggregation takes place as part of the process of classification (or vice versa). For instance, when classifying persons into the category PERSON, the (aggregation-) process of deciding the relevant attributes of the persons takes place at the same time.

3. Database Concepts Described in an Object Oriented Perspective

A database is an information system intended to represent (model) some referent system (see Fig. 1), and this fact is reflected in our definition:

A **database** is

- an information system that
- provides information about the state of another system, the referent system, and the database is designed to be a model of (to represent) the referent system.
- The desired correspondence between actual *database states* and referent system states, at any given time, is obtained through a set of routines reflecting an *update strategy*, and by a set of *mappings* from the database, extracting database state information in the form of *maps*.

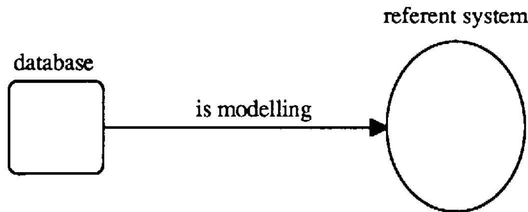

Fig. 1. A database is modelling a referent system.

An example:
A database of the employees in an enterprise is an information system used in personal administration. The enterprise and employees, with their attributes, constitute the referent system. In order to keep the database state in correspondence with the state of the referent system, a change in the referent system (e.g. a new person employed in the enterprise) causes a change in the database (a new employee-object inserted in the database). The change in the database is done in accordance with an update strategy (e.g. once a day, when "required"). Information about the employees (e.g. *name* and *salary*) could be presented as a table (the map) by a mapping from the actual database state.

The structure of the objects is specified by means of category descriptions in a *database schema*. The selected categories and their structure specifications reflect a *perspective* on the referent system (the model perspective).

3.1 Database State - Representative State

A state of a database is expressed by describing:
- The moment at which the state is recorded.
- The *objects* (e.g. employees, departments), the *relationships* (e.g. relationship between an employee and a department), the values of the *attributes* (e.g. *date of birth* of an employee, *name* of a department), and the ongoing *transformations* (actions on the objects - e.g. execution of update-procedures).

The information obtained by using a database does usually not include ongoing transformations. For instance, a database is *not intended to give* information about an ongoing move of an employee from one department in the enterprise to another - the purpose of the database is to give information about which department the employee actually belongs to (and perhaps former departments). This fact is reflected in our definition of *representative states* (see Fig. 2):

> **Representative states** of a database are states concerning the *objects*, *relationships* and *attributes* that have counterparts in corresponding states in the associated *referent system*.

Relationships between objects are expressed either as references or as objects in their own right.
Working files, index tables, the execution of constraint checks, etc. are not part of the representative state.

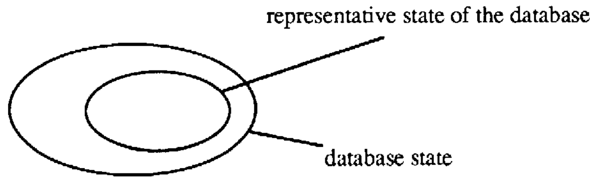

Fig. 2. Representative state of the database as part of the database state.

In a database system, the ongoing actions *per se* are not of interest from an end-user's point of view - it is the results of the actions that are of interest. End-users are interested in representative states *between* transitions.[3] A control system (e.g. a system monitoring a nuclear power plant or an auto-pilot system) is an example where information about ongoing actions is of interest. Similarly, information about ongoing actions of a simulated referent system is central when regarding a simulation system.

A special example of a description of a representative state of a database is a *back-up*. The back-up shows (is a picture of) a database in a certain (representative) state - it preserves that state. A back-up is not itself a database, but it can be used to initialize a database with the same structure as the database for which the back-up was made.

3.2 Update of the Database

In order to provide information about the actual state of the referent system, the database should (in theory) be updated at the same point of time as the changes occur in the referent system. In practise, however, there is often a period of time (a delay) between the change in the referent system and the corresponding update of the database. This phenomenon depends essentially upon the update strategy, which we divide into four main strategies:

1. The update of the database system is executed *before* the corresponding action in the referent system (e.g. room-reservation, flight-reservation).
2. When any change in the referent system occurs, a corresponding change in the database is made immediately (e.g. bank account system).
3. Batch update - the changes in the referent system that have occurred during a certain time interval (since last update), will be correspondingly changed in the database - all at the same time (once a day, once a week, etc.).
4. Update when "required" - the changes are made each time somebody needs the database to get information about the referent system. The time intervals between the updates according to this strategy are thus not equal.

In practise, the update procedure is often a combination of the strategies outlined here.

[3] The ongoing actions are, of course, of interest when regarding *concurrency*. The interleaving of concurrent operations can produce incorrect (not representative) states of the database. In order to prevent this situation, the control mechanism known as *locking*, is introduced.

3.3 Mapping

In order to *present* the information of a database, we introduce the notions of *mappings* and *maps*. A map is a presentation of information and may be given in the forms of screen displays, statistical tables, graphical displays, etc. A map is produced as a result of a mapping from the *database space* to the *presentation space*. The database space is the space of all possible (representative) database states of a certain database, and the presentation space is the space of all possible presentations of a corresponding database.

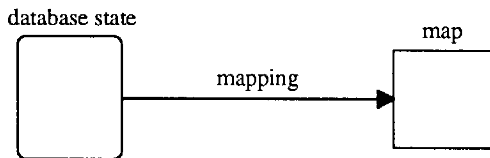

Fig. 3. The mapping.

4. Data Model

The concept data model is frequently used within informatics - in data modelling in general and in the domain of databases in particular. In this domain, the development of the three classical data models for "database design" - the relational data model [*Codd, 70*], the network data model [*DBTG, 71*] and the hierarchical data model (e.g. [*Tsichritzis, Lochovsky, 76*]) - has taken place.

The model concept, as it is used in this context, does not correspond to our definition of the concept. From our model definition, a data model should be a *model* of a part of the world (the referent system). Rather, a data model is a collection of *principles and guidelines for the process of constructing and manipulating model systems* (e.g. databases), and a data model always reflects a perspective on this process. Instead of talking about different data models, one could thus talk about different perspectives - e.g. *the hierarchical perspective, the network perspective, the relational perspective.*

We define the concept as follows:

> A **data model** is characterized by having an *inherent structure* and a set of *tools* and *techniques* used in the process of designing, constructing and manipulating model systems (e.g. databases). The data model reflects a *perspective* on this process.

A model system has a *structure*. Some part of the structure is given by general principles - *independent of the actual referent system being modelled.* We refer to this part of the structure as the *inherent structure.*[4] In fact, it would be more correct to say that a data model supports *general principles for imposing* the inherent

[4] This concept is a modification of the concept *inherent constraint* used in e.g. [*Tsichritzis, Lochovsky, 82*].

structure of model systems. However, we find it more convenient (in accordance with common convention) to talk about *inherent structure of a data model* when discussing data models.

Examples of inherent structure of data models are the representation (structure) in the form of *hierarchies* in the hierarchical data model and in form of *networks* in the network data model. The selected categories and their attributes (including the constraints on these) give structure to a database which is *dependent of the actual referent system*, and such imposed structure is, of course, not based on principles of a data model.

Techniques of a data model could be modelling methodologies as diagrammatic techniques, techniques for schema design, query design, etc. Tools could be languages, concepts, constructs, abstraction mechanisms, automatic design aids, etc.

4.1 Database Language

The language used to describe the referent system and to describe and prescribe the structure and use of a database, we call a *database language* (DBL). A data definition language (DDL), a data manipulation language (DML) and a query language (QL) are subsets of a database language. We refer to DDL, DML and QL as *aspects* of the database language, though they today often are languages in their own right. They are used for defining, manipulating and querying databases. The distinction between the languages (aspects) is not clear. For example, should a declaration of a function be written in DDL or DML? This is one of the reasons why all these languages (aspects) should - in our opinion - be unified into one language[5].
The basic actions or operations (by means of predefined procedures or operators) that can be performed on a database, are:

retrieve (make available objects in the database)
insert (add new objects to the database)
delete (remove objects from the database)
update (change existing values of attributes in objects in the database)

Such operations transform a database state into another database state. If the operation is insert, delete or update (also referred to as *altering operations*), the operation also transforms the *representative state* of the database into another representative state.

The *structure of a database* is the limitations of its set of states and state transitions. The database states and the database state transitions are limited by two aspects of structure: the *permanent structure* and the *transient structure*.

The permanent structure is written in the DDL and DML aspects of the database language and is contained in *database schemas*. The DDL-described structure is the category descriptions of objects and their attributes together with the relationships between the objects. This structure is described in the *basic schema*

[5] SQL (Structured Query language), which has become an ANSI standard for the relational data model, is an example of such a unifying language.

(corresponding to the *conceptual schema* in the "ANSI/SPARC three schema architecture" [*ANSI/SPARC, 75*]). The DML-described structure is contained in *external schemas* (cf. ANSI/SPARC). The main purpose of the external schemas is to make it possible to produce maps adapted to special users (user groups). Another purpose is to protect parts of the database from different user groups. This concerns both updates and maps. A database has *one* associated basic schema, while there (normally) are *several* external schemas. When external schemas are made, the basic schema is used as basis.
The transient structure is provided by the QL aspect of the database language. A query is formed ad hoc - as opposed to a procedure (permanent structure). The QL may, however, specify the invocation of existing permanent structure, described for instance in the DML.

As mentioned, a *data model* is characterized by having an *inherent structure* and a set of *tools* and *techniques* used in the process of designing, constructing and manipulating databases. Another way to express the data model concept, is to regard a data model as a concise reflection of the *perspective* of one or more database languages. This perspective and the perspective reflected in the actual process of *using* the database language (and data model), are reflected in the modelled database:
If a referent system with a specific structure is modelled (e.g. an enterprise with hierarchical structure), it would be convenient, of course, to use a data model with an inherent structure adapted to the structure of the referent system (e.g. the hierarchical data model). In turn, when the data model is selected, a database language, among the set of associated database languages of this data model, must be chosen. The perspective reflected in the database is further specialized by the actual use of the database language (mainly by the DDL aspect) which might be restricted by e.g. economy and time limits.

4.2 Classification of Data Models

Data models can be classified into four categories (cf. [*Brodie, 84*]):

1. Primitive data models (traditional file systems, where objects are represented in records grouped into files, and the operations provided are primitive read and write operations on records).
2. Classical data models (the relational, the network and the hierarchical data model).
3. Semantic data models (data models designed to provide richer and more expressive concepts in which to capture more "meaning" than the classical data models).
4. Special purpose (application oriented) data models (semantic modelling theory applied to particular applications such as office and factory automation, VLSI, CAD/CAM, etc.).

Data models of category two and three concern databases, and they are also referred to as *database models* (e.g. [*Skarra, Zdonik, 87*]). There is, however, no clear distinction between the categories. For example, semantic data models can be used in the domain of databases, and they can be used in other domains (e.g. the *semantic network data model* in the domain of AI).

4.3 Concept Usage

Data models differ to a great extent in terminology. The domain of databases and data models would profit greatly by a standardization of concept usage. This problem concerns, in particular, the concepts *object* and *attribute* which have a lot of synonyms. The most common used synonyms are *record* and *field*. These concepts (and concepts *tuple* and *segment*) are - in our opinion - less general than the concepts object and attribute. *Record* and *field* are only suitable when considering *physical representation* - as opposed to when considering the referent system.
The concept usage in the three classical data models and the two semantic data models (the entity relationship model (ER) [*Chen, 76*], [*Chen, 77*] and the extended relational model RM/T [*Codd, 79*]) is presented in Fig. 4. The concept usage is compared with our proposed terms given in the leftmost column. For relations (associations) between objects we use the term *relationship* (as in ER). A relationship is classified into a *relationship category*.

4.4 A Comparison of some Data Models

We will now briefly compare the three classical data models and the two semantic data models ER and RM/T with respect to concept usage, abstraction mechanisms, inherent structure and specification of attributes.
The use of concepts in the data models differs significantly. This is reflected in Fig. 4.

proposed terms	corresponding terms				
	Entity Relationship	**Relational**	**RM/T**	**Network**	**Hierarchical**
object	entity	tuple, row	entity	record	segment, record
object-category	entity set	heading, relational scheme	entity type	record type	reord type
relationship	relationship	tuple, row	associative entity	set (DBTG-set), set occurence	link
relationship category	relationship set	heading, relational scheme	associative entity type	set, set type (DBTG-set type)	link
attribute	attribute	attribute, column	attribute	data item	field (field occurence)
type-category	value set	domain	domain	?	field

Fig. 4. Concept usage in some specific data models.

Today, the attributes supported are mainly quantities. In future, attributes should also include references and categories (procedures and functions, in particular).

Abstraction Mechanisms

In the data models, there are different constructs for abstracting (*classifying*) objects, relationships and attributes (see the rows *object-category*, *relationship-category* and *type-category* in Fig. 4).

Generalization is supported in the RM/T only.[6] The lack of generalization often results in an unnatural and awkward way of modelling the referent system. This abstraction mechanism is well known from object oriented languages as SIMULA, BETA and Smalltalk.

In all the data models, *aggregation* is supported at one level only - a category can be regarded as an aggregation of its constituent attributes (e.g. PERSON is aggregated from *name*, *address*, etc.). It would be convenient to model object-categories as *containing* other object-categories (*nesting*). For example, an object-category INSTITUTE could be specified as an aggregation of the object-categories STUDENT, LECTURER, MACHINERY, etc. However, in the data models discussed, it is not possible to aggregate object-categories in this way.

Inherent Structure and Relationships

The main difference between data models concerns the inherent structure - mainly in how relationships between objects are expressed.

The ER, the network, and the hierarchical data model have specific constructs for expressing relationships. The ER data model (like the relational data model) supports direct representation of many-to-many relationships, while so-called *virtual* object-categories must be introduced in the network and the hierarchical data model in order to express such many-to-many relationships.

In the relational data model, there is no specific construct for expressing relationships - both objects and relationships are represented as *relations* (perceived by its users as a collection of tables). The users (database designers and application programmers, in particular) are thus not limited by a strict structure (e.g. network and hierarchical structure) when modelling relationships.
There is no difference in the representation of one-to-one, one-to-many or many-to-many relationships in the relational data model, and this data model is the only one supporting direct representation of relationships between *more than two* object-categories. The existence of the associated objects of a relationship must be explicitly checked (cf. the problem of *referential integrity*, e.g. [*Date, 86*]). In the network and hierarchical data model, it is impossible to insert a "child-object" without the existence of a "parent-object". Correspondingly, all the "child-objects" of a relationship are automatically removed when removing the "parent-object".

---

[6] Generalization is, of course, supported in other semantic data models not discussed in this paper (e.g. the *semantic network data model* (the *IS-A* construct)).

The languages of a data model are adapted to the inherent structure. Since the inherent structure of the relational data model is reflected by the fact that the objects and relationships in relational databases are represented as relations (sets), access and manipulation of these databases are *set-oriented.* Relational DBMSs are thus said to have a *set-interface*. In network and hierarchical databases, objects and relationships are accessed and manipulated one at a time. Hierarchical and network DBMSs are thus said to have a "one-record-at-a-time" interface.[7]

Attributes

The data models support quantity attributes (in fact, quantity-variables) only - the ability to specify procedures and functions as attributes in the category-descriptions is not supported. This fact tells us that the object oriented perspective has not (yet) significantly influenced the field of databases and data models.

The ER and the relational data model give the ability to specify user-defined *value sets* (*domains*) - used as type-categories. For example, the type-category AGE could be specified with value set integers from 0 to 150. In our opinion, the "set" concept in ER is not convenient - a set should denote a set (value set or extension of categories). As the concepts are used in ER, however, they denote both the *intention* and the *extension* of categories. The concept *value set* in ER is used to abstract attributes (quantity-variables) with their values. Similar to *entity set* and *relationship set*, the concept *value set* is used both as *category* (*type-category*) - and as *set*. We use *type-category* instead of *value set*, and we use *value set* to denote the set of legal values of the associated attribute.

The network and the hierarchical data model do not support the ability to specify user-defined type-categories - all type-categories are predefined.

5. DBMS

A *database management system* (DBMS) is a package of software which makes it possible to define and manipulate databases in computers. In addition to the implementation of a database language (compiler, interpreter, run-time system, etc.), a DBMS often includes facilities as full screen editor, report generator, *system catalog*, etc.

The simplest example of a DBMS is just a file system in which the database is viewed by the users as a group of files, each file consisting of records with fields. Another example is a relational DBMS as INGRES[8] in which the database is viewed by the users as relational tables. Loosely speaking, the users interact at a higher conceptual level. INGRES is more advanced and has facilities as menu and form driven interface, report generator, system catalog, etc.

---

7 Many *non-relational* DBMSs have (or are currently extended to have) "relational" ("set") front-end. However, problems often occur because of the dependency on the underlying structure.

8 INGRES [*Stonebraker, 76*] is developed at the University of California at Berkeley.

Information about structure of a database is supported by the DBMS by means of the system catalog (or *data dictionary*). In addition to structure information, it contains information as number of objects of different categories, and information not concerning the referent system - such as indexes, update-frequencies, users, access privileges, etc.

A DBMS can be regarded as an *implementation of tools* (e.g. languages) *associated with a data model.* The perspective of the associated data model is thus reflected in the DBMS. The associated data model is also referred to as the *inherent data model* of a DBMS (e.g. [*Tsichritzis, Lochovsky, 82*]). A DBMS might have several associated data models, and it can thus be difficult to classify DBMSs according to the associated data model(s). For example, ADABAS[9] is denoted both as a *network* DBMS (e.g. [*Ullman, 82*]) and as an *inverted list* DBMS[10] (e.g. [*Date, 86*]).

The connections between data model, DBMS and database can be explained with respect to structure. A data model imposes structure on a database via (by means of) a DBMS (see Fig. 5).

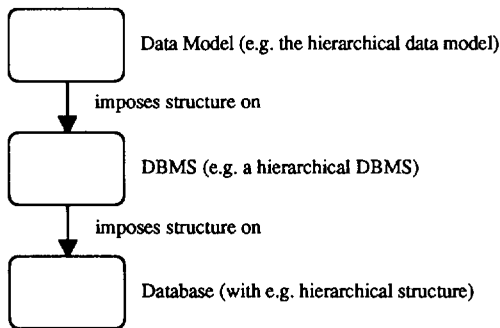

Fig. 5. Structure diagram.

In addition to the implementations of tools associated with a data model, a DBMS includes facilities independent of the associated data model (e.g. tools concerning user-interface).

6. Database Concepts in a Process/Structure Hierarchy

In this section, we describe the development and use of databases in a process/structure hierarchy, and we introduce the notions of "database process", "actors" and "database machine".

9 ADABAS is a DBMS from Software AG.

10 "An inverted list database is similar to a relational database - but a relational database at the low level of abstractions, a level at which the stored tables themselves *and also certain access paths to those stored tables* (in particular, certain indexes) are directly visible to the user" [*Date, 86*].

6.1 The Database Process

In a database process, the *database* is the "part of the world" being developed (cf. the process definition):

A **database process** is a development of a database through state transitions during a time interval.

The *subprocesses* of a database process can be described with respect to how the database is transformed:

1. Subprocesses causing transitions of the representative state of the database - the database is updated (e.g. the address of a person is changed).
2. Subprocesses exclusively producing maps by means of mappings (for instance, the process of making statistical tables concerning the age distribution of persons).

In practice, a subprocess may belong to both categories.

The execution of the database process can be classified into the following categories:

1. *Uni-sequential execution.* All subprocesses are organized in one single sequence (no alternation or concurrency between sequences occurs).
2. *Alternate execution*[11]. Every subprocess is organized (at a given time) in *one* of a finite set of sequences (no concurrency between sequences occurs).
3. *Concurrent execution.* Every subprocess is organized as a finite set of sequences (concurrency between sequences may occur).

6.2 Actors

An **actor** is a participant in a process, transforming the process states through its actions.

A **system development actor** is a participant in a process (the system development process) generating and/or modifying permanent structure for database processes.

Designers and programmers are typical system development actors. A system development actor takes part in the system development process by specifying a referent system and by describing and prescribing permanent structure for a database process (and thereby the permanent structure of the database) in database schemas by means of a database language.

A **database actor** is a participant in a database process, transforming database states through its actions.

11 Alternate execution is described in [*BETA, 85*] by the specific concepts *coroutine sequencing* and *alternation* which denote deterministic and non-deterministic alternate execution, respectively.

End-users are typical database actors. Other information systems (e.g. simulation systems, control systems) interacting with databases, are also examples of database actors.

Database actors take part in the database process by describing and prescribing transient structure for the process (and thereby transient structure of the database) by means of the QL aspect of a database language. The set of possible descriptions and prescriptions is limited by the permanent structure specified by the system development actors.

System development actors, participating in the system development process, modify and produce permanent structure, while database actors, participating in the database process, produce transient structure. The actual occurrences of transient structure are not of interest to a system development actor. The system development actor is, however, interested in creating powerful tools for creation of transient structure by database actors.

A database actor might want transient structure to become permanent (structure). For example, an end-user who repeatedly forms the same query in order to produce a specific map, might want a permanent structure for that mapping.
The end-user might then change his role from database actor to system development actor (the end-user acts as system development actor).

6.3 Database Machine

A **database machine** is a computer that carries out a database process according to (permanent and transient) structure communicated (and/or described) by system development actors and database actors in an associated database language.

The concepts *database*, *database machine*, *database language* and *database process* are interrelated:

To every database, there is an associated database process carried out by an associated database machine and with structure given in an associated database language. To every database machine, there are an associated database, a database process and a database language. To a database language there exists a data model and there may exist a set of (database, database machine) couples.

6.4 Summary of Concepts

Fig. 6 summarizes the connections between some database concepts. The figure shows that both system development actors and database actors use a database language when producing and modifying permanent and transient structure description, respectively. These structure descriptions are used as prescriptions for the database machine which carries out the database process, which, in turn, transforms the database. The system development actors produce and modify structure description in accordance with the actual referent

system being modelled. In addition to the transient structure produced by forming queries, the database actors produce transient structure as results of changes in the referent system (updates).

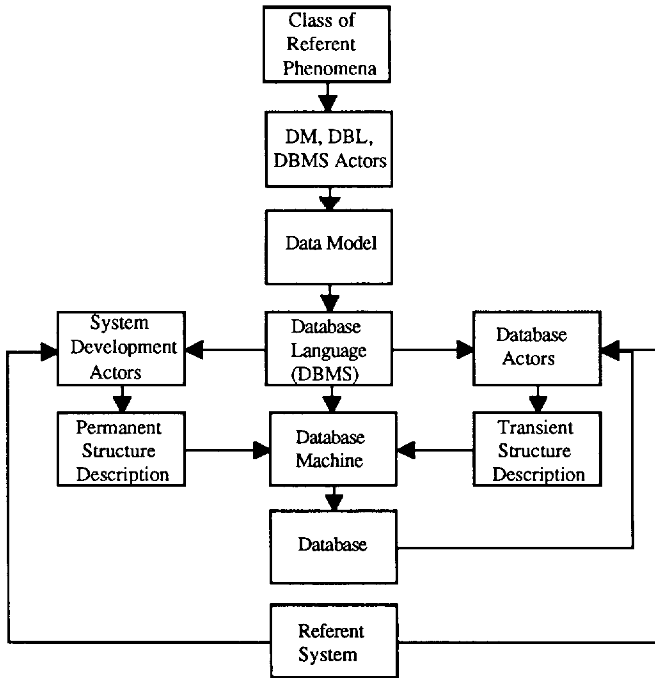

Fig. 6. Some interrelated database concepts.

DBMS (with associated data model and database language), system development and database actors are participants in, and are carrying out, processes. These processes produce structure which, in turn, limits a process (with participant actors) at lower levels. Processes and structures concerning development and use of databases can be described in a hierarchy. When applying this strict actor/process view (perspective), we identify the levels of the following process/-structure hierarchy:

Level 1:

Process: The database process (a set of subprocesses causing state transitions of the database) carried out by a database machine.

Structure: Transient structure (and permanent structure from level 2) limiting the database process.

Level 2:

Process: The user processes by database actors (processes producing transient structure - at level 1).

Structure: Permanent structure limiting the user processes (and database processes).

Level 3:

Process: The system development process by system development actors (processes producing and modifying permanent structure - at level 2).

Structure: Structure of the system development process (DBMS with data model and database language and, e.g., laws, existing knowledge, available resources).

Level 4:

Process: The research and tool development process by DBMS development actors.

Structure: .....

:
:

In general, structure at level n inherits structure at level n+1. For example, the database process is limited by transient structure (at level 1) and by permanent structure (at level 2) and by structure (at level 3) given by the actual DBMS (database language).

Acknowledgement

We would like to thank Kristen Nygaard for valuable comments and discussions, which, in particular, have contributed to the general perspective reflected in this paper. Thanks also to Birger Møller-Pedersen for useful hints and comments.

References

[*BETA, 83a*]: Kristensen, B. B., Madsen, O. L., Møller-Pedersen, B. and Nygaard, K.: "Abstraction Mechanisms in the BETA Programming Language". Proceedings of The Tenth ACM Symposium on Principles of Programming Languages, January 24-26 1983, Austin, Texas.

[*BETA, 83b*]: Kristensen, B. B., Madsen, O. L. and Nygaard, K.: "Syntax Directed Program Modularization". In "Interactive Computing Systems" (Ed. Degano, P. and Sandewall, E.), North-Holland 1983.

[*BETA, 85*]: Kristensen, B. B., Madsen, O. L., Møller-Pedersen, B. and Nygaard, K.: "The BETA Programming Language." Norwegian Computing Center, Report no 805, November 1987.

[*BETA, 87*]: Kristensen, B. B., Madsen, O. L., Møller-Pedersen, B. and Nygaard, K.: "Multisequential Execution in the BETA Programming Language." ACM Sigplan Notices, Vol. 20, No. 4, April 1985.

[*Brodie, 84*]: Brodie, M. L.: "On the Development of Data Models". Topics in Information Systems, On Conceptual Modelling (Perspectives from Artificial Intelligence, Databases, and Programming Languages) 1984 by Springer-Verlag New York Inc. pp. 19-47.

[*Chen, 76*]: Chen, P. P.: "The entity-relationship model: Toward a unified view of data". ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, pp. 9-36.

[*Chen, 77*]: Chen, P. P.: "The entity-relationship model: A basis for the enterprise view of data". Proc. AFIPS NCC No. 46 pp. 77-84.

[*Codd, 70*]: Codd, E. F.: "A Relational Model of Data for Large Shared Data Banks". Communications ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

[*Codd, 79*]: Codd, E. F.: "Extending the data base relational model to capture more meaning." ACM Transactions on Database Systems, Vol. 4, No. 4, December 1979, pp. 397-434.

[*Copeland, Maier, 84*]: Copeland, G. and Maier, D.: "Making Smalltalk a Database System". Proc. ACM Sigmod, Boston, MA, June, 1984.

[*Dahl, Nygaard, 65*]: Dahl, O.-J. and Nygaard, K.: SIMULA - a Language for Programming and Description of Discrete Event Systems. Norwegian Computing Center, Oslo 1965.

[*Dahl, Myhrhaug, Nygaard, 65*]: Dahl, O.-J., Myhrhaug, B. and Nygaard, K.: SIMULA 67 Common Base Language. Norwegian Computing Center, Publ. S-2, Oslo, 1968.

[*Date, 86*]: Date, C. J.: An Introduction to Database Systems. Volume I, fourth edition, Addison-Wesley Publishing Company.

[*DBTG, 71*]: Codasyl Data Base Task Group Report, April 1971.

[*DELTA, 75*]: Holbæk-Hanssen, E., Håndlykken, P. and Nygaard, K.: System Description and the DELTA Language. Norwegian Computing Center, Publ. 523, Oslo 1975.

[*Goldberg, Robson*]: Goldberg, A. and Robson, D.: "Smalltalk-80, the language and its implementation". Addison-Wesley, 1983.

[*Lindsjørn, Sjøberg, 87*]: Lindsjørn, Y. and Sjøberg, D.: "Database Concepts. A Discussion in an Object Oriented Perspective". Master Thesis. University of Oslo, Norway. July, 1987.

[*Maier, Stein, 87*]: Maier, D. and Stein, J.: "Development and Implementation of an Object-Oriented DBMS". Research Directions in Object-Oriented Programming. Shriver, B., Wegner, P. (eds.) The MIT Press. Cambride, Massachusetts. London, England, 1987. pp. 355-392.

[*Nygaard, 86a*]: Nygaard, K.: "Proceedings of the IFIP 10th World Computer Congress". Dublin, Ireland, September 1-5, 1986.

[*Nygaard, 86b*]: Nygaard, K.: "Basic concepts in object oriented programming". (Opening lecture at the Conference on Object Oriented Programming in Wiesbaden, 24-25 September 1985).

[*Skarra, Zdonik, 87*]: Skarra, A. H. and Zdonik, S. B.: "Type Evolution in an Object-Oriented Database". Research Directions in Object-Oriented Programming. Shriver, B., Wegner, P. (eds.) The MIT Press. Cambride, Massachusetts. London, England, 1987. pp. 393-415.

[*Stonebraker, 76*]: Stonebraker, M., Wong, E., Kreps, P. and Held, G.: "The Design and Implementation of INGRES". ACM Transactions on Database Systems, September, 1976.

[*Tsichritzis, Lochovsky, 76*]: Tsichritzis, D. C. and Lochovsky, F. H.: "Hierarchical Data Base Management: A Survey". ACM Comp. Surv. 8, No. 1 (March 1976).

[*Tsichritzis, Lochovsky, 82*]: Tsichritzis, D. C. and Lochovsky, F. H.: Data Models. Prentice-Hall, Englewood Cliffs, N.J., 1982.

[*Ullman, 1982*]: Ullman, J. D.: Principles of Database Systems. Rockville, Md., Computer Science Press, 1982.

[*Zdonik, Wegner, 86*]: Zdonik, S. B. and Wegner, P.: "Language and Methodology for Object Oriented Database Environments". Proc. 19th. Annual Hawaii International Conference on System Sciences, January, 1986.