Chapter 4

Advanced database systems

Essential reading

For "Limitations of the relational model" refer to

"Database Systems: A Practical Approach to Design, Implementation and Management", second edition, by T. Connolly and C. E. Begg, 1999, [ISBN 0-201-34287-1], *Chapter 21* (pp. 728-737).

Alternatively, for a more limited presentation

"An Introduction to Database Systems", sixth edition, by C. J. Date, published by Addison-Wesley, 1995, [ISBN 0-201-82458-2], *Chapter 22 (pp. 630-634).*

For "Deductive databases" refer to

"An Introduction to Database Systems", sixth edition, by C. J. Date, published by Addison-Wesley, 1995, [ISBN 0-201-82458-2], *Appendix C (pp. 775-814).*

For "Object Oriented database systems" refer to

"Database Systems: A Practical Approach to Design, Implementation and Management", second edition, by T. Connolly and C. E. Begg, 1999, [ISBN 0-201-34287-1], *Chapter 21, Chapter 22* (pp. 737-807).

Alternatively, for a less comprehensive introduction, refer to

"An Introduction to Database Systems", sixth edition, by C. J. Date, published by Addison-Wesley, 1995, [ISBN 0-201-82458-2], *Chapter 22, Chapter 23, Chapter 24* (pp. 630– 684).

Connolly and Begg slightly bias their presentation towards object oriented (OO) database systems. They see them as the next stage in the development of database systems (as the relational databases systems were for the previous models).However, they do not say that in a few years time OO database systems will become the standard. They simply envisage a huge development in this area.

Date, on the other hand, has a slightly critical standpoint with respect to object orientation. He affirms that the relational approach is still the best, and, rather than trying to develop other models, it is better to improve and fully exploit the relational model in order to address some of the limitations of existing systems.

For "Object relational database systems" refer to

"Database Systems: A Practical Approach to Design, Implementation and Management", second edition, by T. Connolly and C. E. Begg, 1999, [ISBN 0-201-34287-1], *Chapter 23* (pp. 809-850).

Alternatively, for a less comprehensive introduction, refer to

"An Introduction to Database Systems", sixth edition, by C. J. Date, published by Addison-Wesley, 1995, [ISBN 0-201-82458-2], Chapter 25 (pp. 686-704).

The authors' bias mentioned above is apparent in these chapters too.

Further reading

For a concise description of the topics of this chapter, refer to

"Database Systems Concepts", second edition, by H. F. Korth and A. Silbershatz, published by McGraw-Hill, 1991, [ISBN 0-07-100804-7], *Chapter 13, Chapter 14* (pp. 425-471).

The new approaches to database systems are reflected in the database *programming* languages that are being developed. For an account of the main four approaches to database programming languages (deductive, persistent, functional and object oriented), refer to

"Database Programming Languages", by N. Paton, R. Cooper, H. Williams and P. Trinder, published by Prentice Hall, 1996, [ISBN 0-13-101825-6].

If you are / become interested in one of these areas in particular, the following books provide a more comprehensive description of the subject (the topic of each book is evident from the title).

"Intelligent Database Systems", by E. Bertino, published by Addison Wesley, 1999, [ISBN 0201877368].

"Knowledge Engineering: Unifying Knowledge Base and Database Design", by J. Debenham, published by Springer Verlag, 1998, [ISBN 3540637656].

"Object Database Development: Concepts and Principles", by D. Embley, published by Addison-Wesley, 1997, [ISBN 0201258293].

"Object Databases: An ODMG Approach", by R. Cooper, published by International Thompson Publishing, 1997, [ISBN 1850322945].

Introduction

Database systems developed using the relational model can be viewed as the second generation[1] of database systems. They arose as a solution to the limitations of "navigational" database systems, that constitute the first generation. Systems in the first generation supported complex data structures on disk and allowed application programs navigational access to them. In terms of efficiency they were good, but their primitives were too low level and complex.

Relational database systems overcome the weaknesses of their predecessors. They provide a simple tabular data structure combined with a high level query language. The list below enumerates some of the main strengths of the second generation database systems.

- Relational database systems are developed based on a theoretical model.
- The data model is perfectly suitable for a wide range of practical applications.
- The simplicity of the model allows efficient implementations.
- Relational database systems provide physical program-data independence.

However, as the demand for database systems extended from mainly business applications to many other application areas, the relational model began to show its weaknesses.

This chapter briefly reviews some of the main limitations of the relational model. Then it presents the two main approaches devised to overcome these weaknesses, namely the deductive and the object oriented (OO) approaches. They represent a step towards the third generation of database systems, whose manifesto is presented in the concluding section.

[1] The classification in generations is made in (Beech 1991).

Limitations of the relational model

The relational model has proved appropriate for many real life applications in industry (for inventories, information about employees, departments, ...), business (e.g. banking), education (e.g. libraries, universities), etc. However, with the increased demand for database systems in other application areas the relational model began to manifest its weaknesses. Its unsuitability to these areas varies from minor problems that could be overcome with a little more effort, to a total inadequacy. These areas include:

- computer aided design (CAD) and manufacturing (CAM);
- office information systems (OIS) and multimedia systems;
- geographic information systems (GIS);
- knowledge based systems (KBS);
- computer aided software engineering;
- digital publishing.

I this section we look more closely at the first four.

Computer Aided Design

CAD systems are software systems aimed at assisting the design process in engineering. Application areas include architecture and civil engineering (design of bridges, buildings, etc.) and mechanical and electrical engineering (car industry, aircraft industry, computer industry, etc.). A CAD system, in the main, consists of a drawing tool, assisting the user in elaborating the design and a module that stores the design. Originally, this module was designed and incorporated in the CAD system in an ad-hoc manner. As the design applications became more complex, incorporating millions of parts, the need to implement this module as a "proper" database became apparent.

To identify the requirements for a CAD database, let us consider the design of computers (hardware). Amongst others, a computer contains a set of boards (mother board, memory board, video card, etc.) all interconnected. Each board consists of tens, even hundreds of interconnected electronic components (integrated circuits, transistors, resistors, etc.). The top left part of such a board is illustrated in Figure 1.

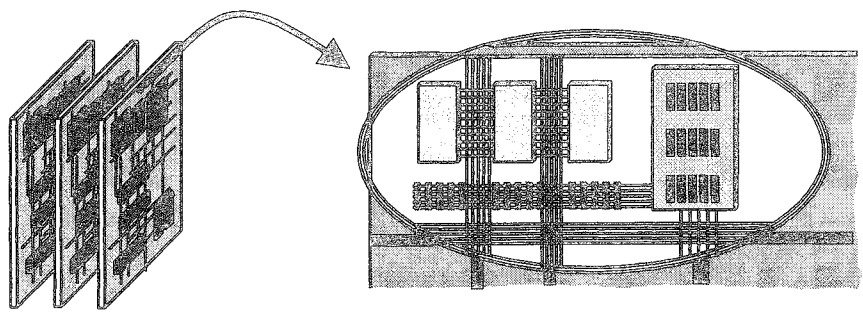

Figure 1: The design of a computer board

Some of the characteristics of this design, and designs from all the areas enumerated above, are:

- The design consists of a large number of components.
- There are small numbers of instances of each type of component. Accordingly, there is a *large number of types* of components in the design.
- An extensive and complex set of relationships (functional, topological and geometric) is established between the design's components.
- As a consequence of the above, the change (update) of a component will have repercussions upon many other parts of the design. The propagation of such an update is, therefore, not an easy process.

- A set of versions is usually associated with a design, each corresponding to a certain design alternatives.

None of these characteristics are easily handled within the relational model.

- The relational model is appropriate for a large number of components. However, it is more suitable for a small number of types, each with many instances, than the other way around.
- Similarly, for the relationships between the components of the system: the relational model is suitable for a small number of types, each being allowed to have a large number of instances.
- The propagation of an update to a component, as described above, will have to be made manually, in the relational model; no support for its automation is provided. This will make the design process very tedious.
- No version control mechanism is provided by relational systems.

In conclusion, the relational model is not suitable for this application area.

Office Information Systems and multimedia systems

Modern systems allow the creation of documents that include formatted text, photographs, diagrams, voice and video sequences. Such documents, apart from the implicit structure imposed by the different types of components, may (and usually do) have an explicit structure, as specified by their creator. For instance, the structure of a world wide web document can be specified by means of the HyperText Markup Language (HTML).

An office, usually, owns a very large number of such documents, therefore they have to be stored in a database. Relational systems are not suitable for such an application.

Almost every document has its own particular structure, so there are almost as many types of documents as documents themselves. Accordingly, in a relational approach, a document can only be stored as a whole, i.e. as a sequence of bytes. This "lump of bytes" has no structure from the point of view of the database system. It is the application program (e.g. a web browser) that identifies (can understand) the structure of a such document. The (relational) database system will not be able to answer queries of the kind:

> For the document "Advert-1.html",get the picture at the top.
> For the document "Advert-2.html", get the second bullet point list from the top.

An extant relational system will only be able to retrieve documents as a whole. Suppose the applications that require parts of documents are remote, connected to the database server via a WAN. Then, the ability of a database system to retrieve just the requested parts can significantly improve performance.

Geographic Information Systems

A similar situation occurs in the area of geographical information systems. Suppose that a database system includes a collection of maps and information about the cities that are shown on each map. If such a database is to be built using a relational system, queries of the kind

> Get the information about population and pollution for the cities within 100 Km of the nuclear plant XXX-1.

cannot be handled.

Knowledge Based Systems

A knowledge based system is a software system that incorporates some knowledge about a certain application area and that is able to explain or to provide advice for a given concrete situation (problem) in the respective application area . For instance, a KBS could be built for diagnosing diseases. Based on the incorporated specialist knowledge, the system can advise a patient, who provides a set of symptoms, about the problem (disease) and possible medication.

This kind of knowledge is expressed in the form of rules, such as:

> If the patient coughs and has sore throat, but has no fever, then the patient has a cold.
> If the patient coughs, has sore throat and fever, then the patient has a chest infection.
> A good treatment for colds is lemon and ginger tea.
> A good treatment for infection is antibiotics.

The number of rules associated with real life applications is, usually, very large. Therefore, they need to be stored in a database.

In a relational approach, such rules would be represented as plain strings (text). It is obvious that, by using such a data structure, all the meaning associated with the rule is lost. Therefore, the relational model is not suitable because it is not able to capture the semantic (meaning) associated with such sentences.

Drawbacks of the relational model

These different application areas illustrated several of the major drawbacks of the relational model and, consequently, of relational database systems. Some of the most prominent ones are (some were not illustrated in the previous examples):

- homogeneous data structure;
- semantic overloading;
- poor representation of real world systems;
- limited operations;
- poor support for integrity constraints;
- difficulty in handling recursive queries;
- type mismatch, usually referred to as impedance mismatch (between the application programs and the database systems).

For further explanations, refer to (Connolly 1999, pp.732-737).

New approaches to database systems, that address some of the drawbacks of the relational database systems, have been proposed. They constitute a step forward towards the third generation of database systems. The most prominent ones are the deductive, object oriented and object relational approaches.

Deductive databases

Informally, in the relational approach, the data base consists of a set of extensionally defined base relations; all tuples for each base relation are explicitly provided.

Executing a query can be regarded as evaluating an expression. The relational operators are applied to the relations involved in the expression and the result is thus obtained. Formally, relational database systems are said to be *model theoretic*. Diagrammatically, this point of view is illustrated in Figure 2.

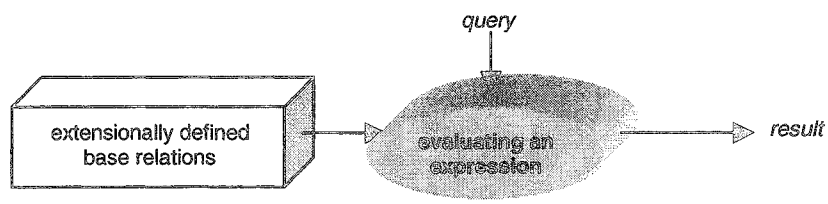

Figure 2: Relational database systems are *model theoretic*

Consider the (toy) relational database illustrated in Figure 3. It contains information about persons, including the parent relationship. Both base relations are defined extensionally. Consider the following query:

Find all fathers (name and address) who have at least one daughter.

One equivalent relational algebra expression involves the base relations and some relational operators[1].

```
( ( Parent JOIN CName = Name ( Person WHERE Sex = F ) ) [PName]
  JOIN PName = Name
  ( Person WHERE Sex = M )
) [Name, Address]
```

The first line of the expression denotes the names of all the persons who have at least one daughter. The third line denotes the male persons. The join of these two represents the fathers who have at least one daughter.

The result is computed by the system by simply performing the specified operations on the base relations.

Person

Name	DoB	Sex	Address
Linda Fox	NULL	F	UB8 3MH
John Fox	NULL	M	UB8 3MH
Mary Fox	NULL	F	1EW 3QA
June Fox	NULL	F	9YT 7NJ
Bill Fox	NULL	M	6HG 6TT
John Hunt	NULL	M	8IO 6GN
Jack Hunt	NULL	M	9PO 6RT
Helen Kent	NULL	F	9YT 7NJ
Dean Kent	NULL	F	9YT 7NJ

Parent

PName	CName
Linda Fox	Mary Fox
Linda Fox	June Fox
Linda Fox	Bill Fox
John Fox	Mary Fox
John Fox	June Fox
John Fox	Bill Fox
Mary Fox	John Hunt
Mary Fox	Jack Hunt
June Fox	Helen Kent
June Fox	Dean Kent

Figure 3: A relational database

Informally, in the deductive database approach, the database consists of a set of facts (equivalent to the base relations in the relational approach) and "rules" (these have no equivalent in the relational approach). They are both expressed in logic; more precisely first order predicate logic or predicate calculus. The set of fact are expressed by means of *ground axioms* and the set of rules are expressed by means of *deductive axioms*.

Informally, ground axioms are statements of facts; for example

```
The person Linda Fox is a female and lives at UB8 3MH
Linda Fox is the parent of Mary Fox
Linda Fox is the parent of June Fox
```

or, more formally,

```
person(Linda-Fox, , F, UB8-3MH)
parent(Linda-Fox, Mary-Fox)
parent(Linda-Fox, June-Fox)
```

Informally, deductive axioms are relationships expressed intensionally. For example, the relation grandparent can be expressed, based on the relation Parent, as (a rule):

```
A person P1 is the grandparent of P2 if there exists a person X such that P1 is
the parent of X and X is the parent of P2.
```

or, more formally,

```
grandparent (P1, P2) IF parent(P1, X) AND parent(X, P2)
```

Executing a query can be regarded as either evaluating an expression purely in terms of the facts of the database (equivalent to the relational approach) or reasoning, i.e. applying the "rules" to the set of facts in order to deduce further facts. As a matter of fact, the former is a particular case of the latter. Formally, deductive database systems

[1] Note that, for readability, the JOIN operator was annotated with the attributes based on which it is to be performed.

are said to be *proof theoretic*. Diagrammatically, this point of view is illustrated in Figure 4.

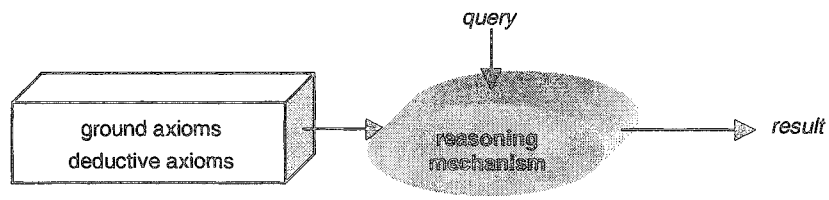

Figure 4: Deductive database systems are *proof theoretic*

If you are not accustomed with propositional and predicate calculus, refer to (Date 1995, pp.778-789). Further, for a more formal description of the concepts of "model-theoretic" and "proof-theoretic", refer to (Date 1995, pp. 189-793).

The rest of this section illustrates the basic functionality of a deductive database system.

The notation conventions used for the rest of this section are as follows. An implication is written / read from right to left and is denoted by the "←" symbol. All constants (excluding the numeric constants) start with a lower case letter. All variables start with an upper case letter. Strings of characters are written between single quotes.

Data definition

The base relations of Figure 3 can be represented as a set of ground axioms.

```
person('Linda Fox', dob(3,4,1938), f, 'UB8 3MH')
person('John Fox, dob(11,3,1936), m, 'UB8 3MH')
person('June Fox', dob(12,11,1960), f, '1EW 3QA')
...
parent('Linda Fox', 'Mary Fox')
parent('Linda Fox', 'June Fox')
parent('Linda Fox', 'Bill Fox')
parent('John Fox', 'Mary Fox')
parent('John Fox', 'June Fox')
...
```

Note that in this formalism the structure of complex objects can be represented. Disregarding this aspect, so far the deductive and the relational models are equivalent

A set of deductive axioms can be added to the above representation. For instance the definitions of the mother, father, brother and sister relations can be given based on the parent relation.

```
mother(X, Y) ←
      parent(X, Y) AND person(X, _, f, _)
father(X, Y) ←
      parent(X, Y) AND person(X, _, m, _)
brother(X, Y) ←
      person(X, _, m, _) AND person (Y, _, m, _) AND
      parent(Z, X) AND parent(Z, Y)
sister(X, Y) ←
      person(X, _, f, _) AND person (Y, _, f, _) AND
      parent(Z, X) AND parent(Z, Y)
```

The same definitions can be expressed in relational systems, via the view mechanism. However, their statement is less elegant than the above.

Activity: Write the view definition, in SQL, for the above relations (mother, father, brother and sister).

One of the strengths of the deductive approach, that does not exist in relational systems, is the possibility of writing recursive definitions. For instance, the ancestor relation can be defined as:

```
ancestor(X, Y) ← parent (X, Y)
ancestor(X, Y) ← parent(X, Z)  AND ancestor (Z, Y)
```

Some of the advantages of using logic (first order predicate logic or predicate calculus, more precisely) for data definition, are

- it provides better support for the representation of real world systems;
- in particular, it provides support for recursive definitions;
- it has a richer semantics, therefore it can be employed in the knowledge representation.

Querying the database

Queries, in the deductive approach, are represented in the same formalism (i.e. predicate calculus formulae). They are expressed as denials (negated predicate calculus formulae).

Two types of queries can be identified:

- yes / no queries;
- find value queries.

The complexity of the answer's computation is low, if the proof procedure uses only ground axioms and high if the proof procedure uses recursive deductive axioms.

Examples for the first type of queries. The following natural language queries

Is Linda Fox the parent of Mary Fox?
Is Linda Fox the mother of Jack Hunt?
Is Linda Fox an ancestor of Dean Kent?

can be expressed as

```
← parent('Linda Fox', 'Mary Fox')
← mother('Linda Fox', 'Jack Hunt')
← ancestor('Linda Fox', 'Dean Kent)
```

For the second type of queries, deductive database systems provide a primitive, "forall", by means of which the whole set of values that satisfy a formula are calculated. For example, the following natural language queries

What is the address of Linda Fox?
Get a parent of Mary Fox.
Who are the parents of Mary Fox?
Who are all the fathers, and what are their addresses, who have at least one daughter?
Who are the ancestors of Dean Kent?

can be expressed as

```
← person('Linda Fox', _, _, Address)
← parent(Parent, 'Mary Fox')
← forall( parent(Parents, 'Mary Fox') )
← forall( person(Father, _, m, Address) AND
          parent(Father, Daughter) AND
          person(Daughter, _, f, _) )
← forall( ancestor(Ancestors, 'Dean Kent') )
```

Some advantages of the deductive approach over the relational approach are apparent

- the same formalism is employed both in data definition and query representation
- the formalism of predicate calculus is more elegant and powerful than that of relational algebra
- in particular, recursive queries can only be handled in the deductive approach.

Updating operations

A deductive database system provides two primitives for updating the database, namely assert and retract. They are equivalent to insert and delete, in the relational approach.

For example, to insert the fact that

David Lloyd, son of Helen Kent, born on 14 March 1967 is a male person and lives at 6TR 8MM

the following statements have to be issued:

```
assert( person('David Lloyd', dob(14, 3, 1967), m, '6TR 8MM') )
assert( parent('Helen Kent', 'David Lloyd') )
```

To delete the fact that Mary Fox is the daughter of Linda and John Fox, the following retract statements have to be issued

```
retract( parent('Linda Fox', 'Mary Fox') )
retract( parent('John Fox', 'Mary Fox') )
```

The main advantage of this approach is that

- the same formalism is used both for data definition and data manipulation

Integrity constraints

In deductive databases

- *rules* – are sentences that define the data and are part of the database
- *integrity constraints* – are sentences which constrain the data and are not part of the database; they are used during the update operations to enforce the integrity of data in the database.

However, in deductive databases, rules and integrity constraints are syntactically indistinguishable. The same expression (predicate calculus formula) can be regarded, in different contexts, as either a rule or as an integrity constraint. It is the intention of the user that defines the type of the expression. If an expression is intended to be a rule, then it should be included in the database. If it is intended as a constraint, then it should be used accordingly (see below).

For example, (supposing people can only have children if they are married!) the relation married-to can be defined based on the parent relation; if two persons have a common child then they are married. This definition can be added to the database in the following form:

```
married-to(X, Y) ← parent(X, Z) AND parent (Y, Z) AND NOT equal(X, Y)
```

Negation poses some important problems for logic based system. It is usually solved in the *closed world assumption*, by implementing *negation as finite failure*. This topic, however, is beyond the scope of this chapter. Therefore the "NOT equal" structure is going to be used without further explanations. (Equality is usually implemented in logic systems.)

This definition, if added to the database, can be used in subsequent queries. For instance, one can find out whether Linda and John Fox are married or not, by posing the query

```
← married-to('John Fox', 'Linda Fox')
```

The same sentence can be interpreted as a constraint: if two persons have a child together then they have to be married. This is expressed exactly as above. However, this time it is a constraint.

```
married-to(X, Y) ← parent(X, Z) AND parent (Y, Z) AND NOT equal(X, Y)
```

For operational reasons, beyond of the scope for this chapter, constraints are usually expressed as denials. So, the above formula has to be translated into a denial, i.e.:

```
← parent(X, Z) AND parent (Y, Z) AND NOT equal(X, Y)
  AND NOT married-to(X, Y)
```

which translates as "It is not true that there are two different persons that have a child together and they are not married".

Suppose that the database, apart from the ground axioms defined at the beginning at the data definition section (i.e., "person" and "parent"), comprises another set of ground axioms, based on the "married-to" predicate, as below:

```
married-to('Linda fox', 'John Fox')
...
```

The above can be used to enforce the condition that whenever two persons become the parents of a child, the "married-to" ground axiom for these persons exists in the database.

There are various mechanisms used for integrity constraint enforcement. We will look at two simple ones.

A simple checking mechanism can be implemented at the time of update. For instance the "make-parent(X, Y)" predicate denotes the fact that X is to be made parent of Y. This fact, "parent(X, Y)", should be asserted only if

(1) it does not already exist in the database as a ground axiom, and
(2) there is no other person Z who has Y as a child and is not married to X.

The definition of "make-parent" is the following:

```
make-parent(X, Y) ← parent (X, Y)
make-parent(X, Y) ← NOT parent (X, Y) AND
                    NOT (parent(Z, Y) AND NOT married-to(X, Z)) AND
                    assert( parent(X, Y) )
```

The constraint checking mechanism can automatically change the database, at the time of the update, in order to maintain the integrity of data. Such an approach is described below:

```
make-parent(X, Y) ← parent (X, Y)
make-parent(X, Y) ← NOT parent (X, Y) AND
                    parent(Z, Y) AND                    (1)
                    NOT married-to(X, Z)) AND           (2)
                    assert( married-to(X, Z) ) AND      (3)
                    assert( parent(X, Y) )
make-parent(X, Y) ← NOT parent (X, Y) AND
                    NOT (parent(Z, Y) AND NOT married-to(X, Z)) AND
                    assert( parent(X, Y) )
```

The definition in the middle states that (1) if there is a person Z who's child is Y and (2) this person is not married to X, then (3) automatically it becomes married to X.

In conclusion:

- deductive database systems provide better support for integrity constraints;
- integrity constraints are based on the same formalism as the one used for data definition and data manipulation.

Conclusions

A deductive database system is characterised by the fact that:

- it supports the definition of both facts (extensionally defined relations) and rules (intensionally defined relation);
  - more precisely, it uses a subset of first order predicate logic (or predicate calculus) for the definition of facts (ground axioms) and rules (deductive axioms);
- it provides a reasoning mechanism;
  - more precisely, it implements a very powerful rule of inference for predicate calculus, namely resolution;
- it uses the same formalism for data definition, data manipulation and the integrity constraint definition, namely a subset of predicate calculus.

Some of the most notable advantages of deductive database systems are:

- it is a "natural" extension to the relational model that represents the standard at the moment;
- it is founded on a sound theoretical model;

- it provides a "natural" formalism to represent and manipulate data and knowledge;
- it uses the same formalism for representing the structure of data, operations on data and integrity constraint imposed on data; in short, it provides representational uniformity;
- it provides operational uniformity, in that different operational problems (such as query optimisation and integrity constraint application) are tackled within the same theoretical model;
- it provides scope for modelling semantic extensions (such as type hierarchies and events).

These advantages do not come for free. Two cardinal problems that obstruct the deductive database systems from becoming widely used are:

- efficiency – due to slowness of the reasoning procedure when applied to large amounts of data;
- persistency – solutions for feasible implementations of persistent logic programming languages are still to be found.

This section only provided a very brief introduction to the topic. Moreover, knowledge of logic programming was assumed in the presentation. To get a better understanding of the topic, refer to the recommended literature. A concise but sufficiently comprehensive introduction to deductive database systems is provided in (Paton 1996), Chapter 2.

OO database systems

Object orientation (OO) is an approach to software development within which some of the "classic" problems of software engineering can be solved. The main advantages of this approach are:

- the employment of *modelling concepts* (such as object, class, message) that have a very close correspondence to components of real life systems; subsequently, these concepts can be animated (i.e. they are implemented in OO programming languages);
- modularity is an inherent characteristic of the approach (in other words, OO systems are inherently modular[1]);
- the support for component development through reusable components (libraries of objects).

The attempt to combine the OO approach with the database approach in what is called Object Oriented database systems was made with a view to eliminating the separation that previously existed between

- the static aspects of information processing, reflected in database systems, and
- the dynamic aspects of information processing, reflected in application programs.

The removal of this division means the removal of some major drawbacks associated with traditional database systems (refer to the first section of this chapter).

OO database systems is an area that has generated a lot of interest in recent years. The present market for it is 3% of the overall database market, but it is expected to grow at a pace of over 50% per year, faster than the total database market (Connolly 1999). The incorporation of the OO approach in the Web technology (for instance, through Java) worked in its favour.

---

[1] This statement is too strong. It is possible to develop OO systems with a low degree of modularity (by bad practice). However, the support for (enforcement of) modular development is far better in the OO approach than in other approaches. Therefore, the above statement can be accepted.

This section presents an overview of the area of OO database systems.

OO concepts

Before talking about OO database systems you should be familiar with the main concepts that are used in object orientation. Since they are not, strictly speaking, a database issue, they are only enumerated in this section. The "Software Engineering and Development" module deals with this topic. However, you can also refer to (Connolly 1999, pp.737-749) or (Date 1995, pp. 630-650).

The aspects that you should (re)consider are:

- abstraction, encapsulation and information hiding
- objects and attributes; object identity;
- methods and messages;
- classes and instances;
- class hierarchies and inheritance;
- polymorphism and dynamic binding.

OO database languages

As described earlier, the development of software applications, in a large number of application areas, requires development systems that support both database and traditional (characteristic to the third generation) programming language capabilities.

Traditionally, the features that are supported within one approach are not supported within the other, as illustrated in Figure 5.

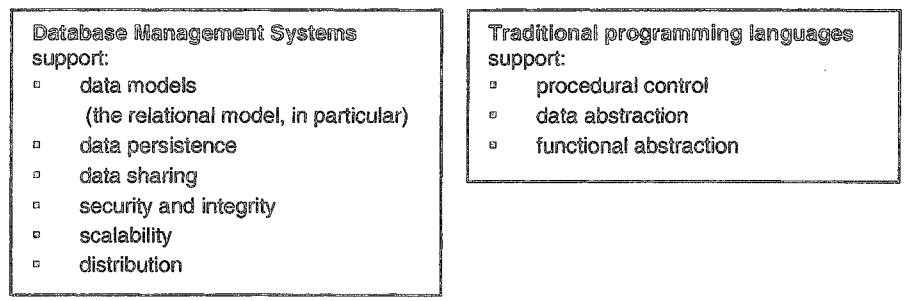

Figure 5: Features of database management systems and of programming languages

The development of a software application often requires features from both groups. Accordingly, parts of the applications would be developed by means of DBMS, whereas the other parts would be developed be means of a traditional programming language. To make the application working, the two parts have to be put together.

The customary approach to resolving this separation was (still is) to *embed* a database language into a traditional programming language (referred to as *host* language). For instance, SQL is embedded in C, Pascal, Ada and Fortran. The embedding procedure is based on the communication between the host language and the DBMS's routines. This communication is achieved by means of a set of variables in the host language plus calls to the DBMS's routines; a pre-processor translates the embedded SQL statements into calls to the routines provided by the DBMS.

This approach presents some major drawbacks:

- impedance mismatch
  - The DBMS and the programming language provide different data types and data structures. Therefore, the programmer has to write code to translate between the host language's data model and the DBMS's data model.
- additional type checking

- While manipulated in the host language the data objects are strongly typed. This typing is lost by storing the data in the database (e.g. data structures are flattened). Type checks must be performed, whenever this data is retrieved from the database. This is because applications written in other programming languages (that do not provide strong typing) might have accessed this data.

- the decision to read and write on persistent storage is the programmer's responsibility (is not transparent).

A solution to the above drawbacks can be based on the single level storage model for accessing data objects (as opposed to the two level storage model that was traditionally used). For further details about this issue refer to (Connolly 1999, 762-764).

The one level storage model can be implemented in the OO approach. Therefore, in the OO approach, a *database programming language* can be devised, that addresses[1] the above drawbacks. Possible strategies for developing such a language are:

- extend an existing OO language with database capabilities;
- provide OO DBMS libraries;
- embed OO database constructs into a conventional language (i.e. that does not support object orientation);
- extend an existing database language with OO capabilities (and increase its computational power, i.e. make it computationally complete); the SQL3 language, expected to be release soon, has been developed using this approach;
- develop a new language from scratch.

The main problem associated with the development of such a language is the provision of transparent persistency. This means that the user does not need to know where the data is, whether in the internal memory or on persistent storage. The user simply deals with data. It is the responsibility of the system to manipulate the data between the permanent and temporary storage; moreover, this has to be done efficiently and without loss or errors. A solution to the problem of transparent persistency is called a *persistency scheme*. For an overview of possible persistency schemes refer to (Connolly 1999, pp.765-769).

OO database management systems

There is no single definition to what an OO DBMS is. Researchers and developers view the concept of OO DBMS from different perspectives.

Kim (Kim 1991) provides the following definition

***Definition:** An object oriented data model (OODM) is a logical data model that captures the semantics of objects supported in OO programming languages. An object oriented database (OODB) is a* persistent *and* sharable *collection of objects defined by an OODM. An object oriented database management system (OOBDMS) is the manager of an OODB.*

This definition reflects the fact that there is no unique object oriented data model (hence, the phrase "the semantics of objects supported in OO programming languages". Other definitions are provided in terms of the features that an OO DBMS should support.

Zdonik and Maier (Zdonic 1990) provide a minimum set of requirements for an OO DBMS:

- database functionality
- support for object identity
- provide encapsulation

[1] This does not mean that these drawbacks are totally eliminated.

- support for objects with complex states.

Khoshafian and Abnous (Khoshafian 1990) define:

Definition: Object Orientation means support for abstract data types, inheritance and object identity. OO DBMS means object orientation plus database capabilities.

Parsaye et al. (Parsaye 1989) require an OO DBMS to (1) be an object oriented systems and (2) to provide the following features:

- provision of a high level query language with optimisation capabilities in the underlying system;
- support for persistence and atomic transactions (concurrency and recovery control);
- support for complex object storage, indexes and access methods for fast and efficient retrieval.

An attempt to standardise the object oriented approach is currently being made by the Object Management Group (OMG). This is an international non-profit making consortium that includes over 700 organisations, virtually all major platform and software vendors, such as IBM, Sun, Dec, Microsoft and Apple (Connolly 1999). All the members agreed to work together to develop standards acceptable to all. One of the major areas of standardisation is the *Object Model* (OM). For details, refer to (Connolly 1999, pp.789-792). Based on the OM, the Object Database Management Group (ODMG) defined a "standard" for OO databases.

The ODMG includes some important vendors of OO DBMS. They have defined the standard for the Object Oriented Data Model (OODM). This standards consists of:

- an object model (a superset of the OM);
- a data definition language;
- a data manipulation language;

and is described in (Connolly 1999, pp.793-805).

We conclude this section with an enumeration of the advantages and disadvantages of OO DBMS.

Advantages:

- enriched modelling capabilities;
- extensibility;
- removal of impedance mismatch;
- more expressive query language;
- support for schema evolution;
- support for long duration transactions;
- improved performance.

Disadvantages:

- lack of a universal data model and lack of standards (efforts are made, though, for standardisation – refer to the previous paragraphs);
- lack of experience (it is still a relatively new area)
- query optimisation compromises encapsulation;
- locking at object level can have repercussions on performance;
- high complexity;

- lack of support for views;
- lack of support for security.

Object relational database systems

Object relational database systems combine, as the name suggests, principles and techniques from both purist approaches (i.e., relational and OO). The interested reader can refer to either (Connolly 1999) Chapter 23 or to (Date 1995) Chapter 25.

Third generation database systems

We conclude the section about advanced database systems with a summary of the features that were proposed by the Committee for Advance DBMS function (CADF) for third generation database systems. This list is known as The Third Generation Database Systems Manifesto.

- Third generation database systems must subsume second generation database systems.
- The features that a third generation database system *must* have include:
  - rich type system;
  - rules (triggers, constraints);
  - updateable views;
  - access to the database through a non-procedural, high level language;
  - the DBMS must be accessible from high level languages;
- Extra features, that *would be nice to have*, include:
  - inheritance;
  - functions (database procedures, methods);
  - versions;
  - long and nested transactions;
  - schema evolution.

For further details refer to (Connolly 1999, p. 814). This manifesto is merely a proposal of a group of people; some will strongly adhere to it, some will disagree. However, it gives you an idea of what is to be expected from the new generation of database systems and, accordingly, what are the main issues associated with their development.

Learning outcomes

On completion of this chapter you should be able to:

- discuss the limitations of the relational model with respect to new areas of development for database systems;
- explain the deductive database approach;
  - explain the major differences between the relational model and the deductive model;
  - describe the main components of a deductive database language using first order predicate logic (data definition, querying the database, updating the database, integrity constraint definition and mechanism for integrity constraint enforcement);
- explain the OO database approach;
  - discuss issues related to OO database languages;
  - define OO DBMSs and present their advantages and disadvantages;
  - present the Object Oriented Database Model as defined by the Object Database Management Group.

References

D. Beech, P. Bernstein, M. Brodie, M. Carey, B. Lindsay, L. Rowe and M. Stonebracker, "Third generation Database System manifesto", in Object Oriented Databases: Analysis Design and Construction, (DS-4) (R.A. Meersman et.al., eds.), North Holland, 1991.

W. Kim, "Object Oriented Database Systems: Strengths and Weaknesses", Journal of Object Oriented Programming, 4(4), 21-29, 1991.

S. Khoshafian and R. Abnous, "Object Orientation: Concepts, languages, Databases and Users, John Wiley, 1990.

K. Parsay, M. Chignell, S. Khoshafian and H. Wong, "Intelligent Databases", John Wiley, 1989.

S. Zdonik and D. Maier (eds), "Fundamentals of Object Oriented Databases in Readings". In Object Oriented Database Systems, Morgan and Kaufmann, pp. 1-30, 1990.