

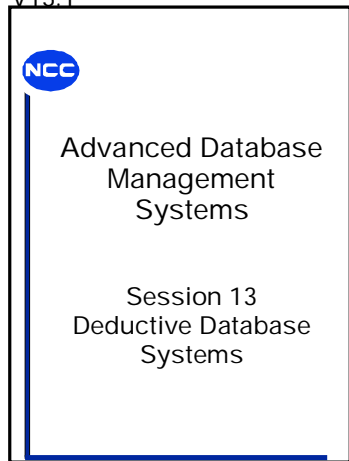
Session 13

Deductive Database Systems

1 Introduction

V13.1

(5 minutes)



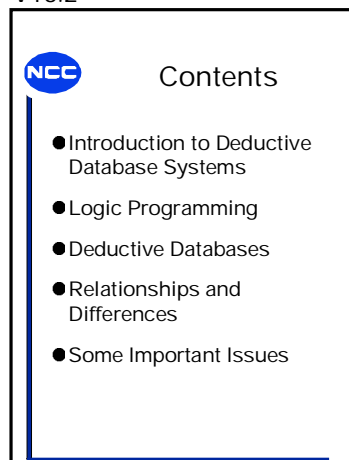
It will not be possible to cover this new research topic in detail within the allocated session time. What this session should aim to achieve is to:

- present a brief introduction to the topic;
- describe the inter-relations between Deductive Database Systems (DDBs), relational database systems and logic programming;
- discuss some important issues involved in the design and implementation of deductive database systems.

Inform students that a handout containing a full set of visuals will be provided to them at the end of this lecture.

1.1 Summary of Topics to be Covered

V13.2



The topics detailed in the visual will be discussed during this session.

2 Introduction to Deductive Database Systems

(10 minutes)

The growing research interest for developing deductive database systems has reflected a general recognition that traditional database systems may not be adequate for some knowledge based applications which require deductive functionality.

A deductive database system is essentially the outcome of applying mathematics logic to database management, where logic is used for both data representation and data manipulation. (Bell et al).

2.1 Description

V13. 3

NCC

Description

- DDB systems are able to make **deductions** from known **facts** and **rules** when processing user queries
 - in other words, they have limited reasoning ability
- Intuitively a DDB system can be conceived as a database capable of deducing additional new data from data that is explicitly represented (facts), by applying specified rules of inference to those facts

- The development of deductive database (DDB) systems has brought together two branches of computer science:
 - logic programming;
 - relational database technology.
- DDB systems are able to make deduction from known *facts* and *rules* when processing user queries. In other words, they have limited *reasoning ability*.
- Intuitively a DDB system can be conceived as a database capable of *deducing* additional new data from data that is explicitly represented (called *facts*), by applying specified *rules of inference* to those facts.

3 Logic Programming - A PROLOG Example

(25 minutes)

V13. 4

NCC

Logic Programming

- A PROLOG Example

```

grandparent(x,y) :- parent(x,z), parent(z,y) (1)
parent(x,y) :- mother(x,y) (2)
parent(x,y) :- father(x,y) (3)
ancestor(x,y) :- ancestor(x,z), parent(z,y) (4)
ancestor(x,y) :- parent(x,y) (5)
father(fred, mary) (6)
father(george, james) (7)
father(john, fred) (8)
mother(sue, mary) (9)
mother(jane, sue) (10)
mother(liz, fred) (11)
mother(sue, james) (12)

```

Most of the deductive database systems use logic programming for logic deduction and an existing relational database management system as a back-end server [Bell et al].

We briefly introduce some aspects of logic programming which are important for deductive database systems. As PROLOG is one of the most commonly used logic programming language, we use it in the examples shown in Visual V13. 4, repeated in Handout 13.1.

Issue Handout 13.1 which also includes the explanation given below.

Explanation:

For example: $\text{grandparent}(x,y) :- \text{parent}(x,z), \text{parent}(z,y)$


The “declarative” meaning of the above clause is:

“for all x,y,z , if x is a parent of z and z is a parent of y , then x is a grandparent of y .”

It is important to note that all variables which appear in a clause are universally quantified at the front (see above example: “for all x,y,z ...”).

3.1 Some Definitions (Lloyd)

V13.5




Some Definitions

- Program - a collection of clauses
- Atom - a predicate followed by some arguments - either variable or constant
- $:-$ - a logical implication
- Head - single atom on the left of the $:-$
- Body - the atoms on the right of the $:-$
- $,$ between atoms means a logical conjunction (AND)

- *Program* - a collection of (program) *clauses*, each consisting of *atom(s)*;
- *Atom* - a *predicate* followed by some *arguments*, which are either variables, or constants;
- $:-$ - ordinary logical implication (\leftarrow);
- *Head* of the clause - the single atom on the left of the $:-$;
- *Body* of the clause - the atoms on the right of the $:-$;
- “,” between atoms stands for logical conjunction (AND).

3.2 Key Points

V13.6



Key Points

- A PROLOG program is a collection of facts and rules
- Facts are clauses with empty body
 - they are **unconditional**
- Rules are clauses with non-empty body
 - they are **conditional**
- Each clause is a shorthand for a first order logic

- A PROLOG program is a collection of *facts* and *rules*.
- *Facts* are clauses with *empty* body. They are *unconditional* and are simply facts.
- *Rules* are clauses with *non-empty* body. They are *conditional*, meaning that *something holds if something else holds*.
- Each clause is a shorthand for a first order logic which is used to express both facts and rules.

3.3 Query Expression (Lloyd)

V13.7

NCC Query Expression in PROLOG

- A query can be expressed in PROLOG as a clause with a body, but with no head
 - a “?” is placed at the beginning of the clause
- Arguments in a query which are constants are used for input to the program, and arguments which are variables are used for output
- Some queries have no variables
 - they only require a yes/no answer

Issue Handout 13.2 which provides the information and examples given below.

- A query can be expressed in PROLOG. It is expressed as a clause with a body, but with no head. A “?” is placed at the beginning of the clause. For example:

Find out who is the father of fred:

`?:- father(x, fred)` “find an x such that x is the father of fred”

The interpreter would respond with the answer: $x=\text{john}$

Many queries have more than one answer.

`?:- mother(sue, x)` “find all x such that sue is the mother of x ”

Answer: $x=\text{mary}$ and $x=\text{james}$

- Arguments in a query which are constants are used for *input* to the program, and arguments which are variables are used for *output*.
- Some queries have no variables. They only require a *yes/no* answer. For example:

`? :-parent(fred, mary)` answer: yes (applying clauses 6,3)
`? :-parent(x,y)` answer: 7 pairs (applying clauses 6-12, 2-3)

`? :-mother(sue,x), father(george,x)` - query with more than one atom
answer: james (applying clauses 9,12,7)

`? :- x:father(x,y)` “find all those persons who are fathers”
 Placing “ x .” before “ $\text{father}(x,y)$ ” will now return only the value for x , not the corresponding y
answer: $x=\text{fred}$, $x=\text{george}$, $x=\text{john}$
 (applying clauses 6-8)

`? :- grandparent(john,x)` answer: $x=\text{mary}$ (applying clauses 6,8,3,1)

- When answering a query, what the PROLOG interpreter does is to prove a query using the program clauses. In theorem proving terminology, each step in the query answering process is a *deduction* using the *resolution* inference rule. The major part of resolution is a pattern matching process called *unification*. During unification, variables become bound to constants and other variables. This is how an answer gets constructed.

3.4 Expressing Data in Relational Databases

V13.8

NCC Expressing Data in Relational Databases - 1

- A relational database consists solely of facts
 - Example - The supplier-part relational databases (Date):
 - $s(sno, sname, status, city)$
 - $p(pno, pname, colour, weight, city)$
 - $sp(sno, pno, qty)$
- Instead of using tables, the tuples of the relations can be expressed like facts in a PROLOG program

Issue Handout 13.3 which provides the information and the examples given in Visuals V13.8, V13. 9, V13. 10 and V13. 11.

A relational database can be viewed as a special type of PROLOG program, which consists solely of facts. For example:

The supplier-part relational databases (Date):

$s(sno, sname, status, city)$

$p(pno, pname, colour, weight, city)$

$sp(sno, pno, qty)$

Instead of using tables, the tuples of the relations can be expressed like facts in a PROLOG program.

The following visuals show examples of:

- Supplier relations.
- Part relation p .
- Supplier-Part relation sp .

V13. 9

NCC Expressing Data in Relational Databases - 2

- Example: Supplier relations
 - $s(s1, smith, 20, london)$
 - $s(s2, jones, 10, paris)$
 - $s(s3, blake, 30, paris)$
 - $s(s4, clark, 20, london)$
 - $s(s5, adams, 30, athens)$

V13. 10

NCC Expressing Data in Relational Databases - 3

- Example: Part relation p
 - $p(p1, nut, red, 12, london)$
 - $p(p2, bolt, green, 17, paris)$
 - $p(p3, screw, blue, 17, rome)$
 - $p(p4, screw, red, 14, london)$
 - $p(p5, cam, blue, 12, paris)$
 - $p(p6, cog, red, 19, london)$

V13. 11

NCC Expressing Data in Relational Databases - 4

- Example: Supplier-Part relation sp
 - $sp(s1, p1, 300)$
 - $sp(s1, p2, 200)$
 - $sp(s1, p3, 400)$
 - $sp(s1, p4, 200)$
 - $sp(s1, p5, 100)$
 - $sp(s1, p6, 100)$
 - $sp(s2, p1, 300)$
 - $sp(s2, p2, 400)$
 - $sp(s3, p2, 200)$
 - $sp(s4, p2, 200)$
 - $sp(s4, p4, 300)$
 - $sp(s4, p5, 400)$


4 Deductive Databases (A More Formal Description)

(5 minutes)

The realisation of the advantages of logic programming and relational database technology has led to the development of DDBs.

- Result of applying mathematical logic to relational database management, where logic is used for both data representation and data manipulation.
- DDBs contain not only facts, but also general rules.

V13. 12



Deductive Database Systems

A more formal description (Date)

- A deductive database system supports the proof-theoretic view of database and in particular is capable of deducing additional information from the extensional database by applying inferential (i.e., deductive) rules that are stored in intensional database. A deductive database system will almost certainly support recursive rules and so perform recursive query processing.

The physical organisation of a deductive database consists of two parts:


- Extensional database.
- Intensional database.

“A deductive database system supports the proof-theoretic view of database and in particular is capable of deducing additional information from the extensional database by applying inferential (i.e., deductive) rules that are stored in intensional database. A deductive database system will almost certainly support recursive rules and so perform recursive query processing.” (Date).

5 Relationships and Differences

(20 minutes)

V13. 13



Relationships and Differences

- A deductive database system has strong links with first order logic, logic programming, theorem proving and relational database theory
 - Deductive database systems and Logic Programming
 - Deductive database systems and relational database systems

A deductive database system has strong links with:

- first order logic;
- logic programming;
- theorem proving and relational database theory.

5.1 Deductive Databases and Logic Programming

Deductive database systems and logic programs both rely on the expressive power of logic programming. There are however differences between them. The main differences are:

- Logic programs generally have more rules than facts, so are usually small enough to be kept in memory.
- Apart from the embedded integrity constraints, deductive database systems normally have only a small number of rules but a very large number of facts, so they must be kept on disk and only those parts required to answer the current query are read into memory. (This requirement on storage and file management in a DDB is similar to that in relational databases.)

5.2 Deductive Databases and Relational Databases

Relational database systems contain only facts, hence the lack of deduction facilities in their implementation.

Deductive database systems are more powerful than relational database systems by applying rules to facts. They can not only retrieve known facts, but also deduce new virtual data using rules which can be recursive.

5.3 Transformation: From RDBs to DDBs

V13. 14

NCC From RDBs to DDBs - Examples

The supplier-part relational database could be made into a DDB by **adding rules**:

For example:

A london-supplier is any supplier based in london

```
london-s(sno,sname,status):-
s(sno,sname,status,london)
```

A supplier is a major supplier if it supplies more than 300 of part number p1

```
major-s(sno):-sp(sno,p1,qty), qty>=300
```

These rules in this example implicitly define two **new relations** ("virtual" relations):

```
london-s(sno, sname, status)
major-s(sno)
```

Now the supplier-part relational database could be made into a DDB by *adding rules*.

- These rules in this example implicitly define two new relations:

london-s(sno, sname, status)

major-s(sno)


- Such relations are *virtual* relations because when a tuple from the relation is requested, it must be *computed* rather than *retrieved*. This is a very similar idea to *views* in a relational database.

- One can add *views* to a DDB by adding rules. But rules are more powerful than views in the following respects:
 - Rules are at the heart of a DDB.
 - Rules become an essential part of data modelling, providing the flexibility of defining a relation partly by some rules and partly by some facts.
 - Rules can be *recursive* (see *Clause 4 of Visual V13.4*).

6 Important Issues

V13. 15

(20 minutes)



Important Issues


- Deductive query language
- Recursive query processing
- Query optimisation
- Integrity constraints

There are many issues involved in the design and implementation of a deductive database systems.

Some of the most important ones are listed in the visual and discussed in the following sub-sections.

6.1 Deductive Query Language

V13. 16



SQL Query - Example

Find the names of suppliers which supply part p2:

```
select sname
from s, sp
where s.sno = sp.sno
and sp.pno = 'p2'
```

is translated into (approximately) the PROLOG query:


```
? :- sname: sp(sno, 'p2', qty), s(sno, sname, status, city)
```

A deductive database system requires a user query language in which the deductive rules can be formulated. The language should also support recursive rules, and hence recursive queries.

The best known example of deductive query language is called Datalog, which is a variant of PROLOG.

6.2 Query Optimisation

V13. 17



Query Optimisation - Example

? :- y: sp(x, 'p2', u), s(x, y, z, w)

- 1st subquery
 - the second argument is given, so only a few pairs of values for x and u are retrieved
- 2nd subquery
 - the first argument is matched against x from the first subquery, so again only a few relevant facts are examined
- To compare with the above, the following evaluation reverses the order of the subqueries:

? :- y: s(x, y, z, w), sp(x, 'p2', u)
- No argument is specified in the first subquery, so every fact about s is retrieved!

As in a relational database system, query optimisation is also very important for a deductive database system where an appropriate *order* for solving subqueries should be chosen so that the total cost of answering the query is minimised.

6.3 Recursive Query Processing

This involves evaluation of queries whose definition is intrinsically recursive (**Date**).

Conventional database languages (for example, SQL) are too weak to express recursion effectively. Deductive database systems allow recursive information to be explicitly specified in databases by applying formal logic.

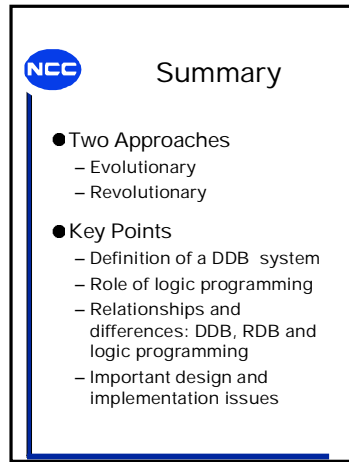
6.4 Integrity Constraints

They can also be included in a deductive database. These constraints are expressed as rules.

7 Summary

V13.18

(5 minutes)



There have been two approaches of developing deductive database systems (Beynon-Davies):

- The *evolution* approach which involves melting together exiting technology or extending existing technology. or example, building a PROLOG system on top of a relational DBMS.
- The *revolutionary* approach which proposes new architectures. For example, Datalog is a proposal which unifies feathurs of logic programming with databases.

Many concepts have been introduced in this session, but the emphasis should be focused on the following *key points*:

- Definition of a deductive database system.
- The role of logic programming in deductive database systems.
- Relationships and differences between logic programming, relational database systems and deductive database systems.
- Important design and implementation issues.

Students are advised to consult the following books *and* relevant research papers in academic journals on the subject:

- C J Date, *An Introduction to Database Systems*, 7th edition, Addison Wesley, 2000.
- R Elmasri & S B Navathe, *Fundamentals of Database Systems*, 3rd edition, Addison Wesley, 1997.

Handout 13.1 – A PROLOG Example

Example:

grandparent(x,y) :- parent(x,z), parent(z,y) (1)

parent(x,y) :- mother(x,y) (2)

parent(x,y) :- father(x,y) (3)

ancestor(x,y) :- ancestor(x,z), parent(z,y) (4)

ancestor(x,y) :- parent(x,y) (5)

father(fred, mary) (6)

father(george, james) (7)

father(john, fred) (8)

mother(sue, mary) (9)

mother(jane, sue) (10)

mother(liz, fred) (11)

mother(sue, james) (12)

Explanation:

For example: *grandparent(x,y) :- parent(x,z), parent(z,y)*

The “declarative” meaning of the above clause is:

“for all x,y,z, if x is a parent of z and z is a parent of y, then x is a grandparent of y.”

It is important to note that all variables which appear in a clause are universally quantified at the front (see above example: “for all x,y,z...”).

Handout 13.2 – Query Expression (Lloyd) – Example

- A query can be expressed in PROLOG. It is expressed as a clause with a body, but with no head. A “?” is placed at the beginning of the clause. For example:

Find out who is the father of fred:

`?:- father(x, fred)` “find an x such that x is the father of fred”

The interpreter would respond with the answer: x=john

Many queries have more than one answer.

`?:- mother(sue, x)` “find all x such that sue is the mother of x”

Answer: x=mary and x=james

- Arguments in a query which are constants are used for *input* to the program, and arguments which are variables are used for *output*.
- Some queries have no variables. They only require a *yes/no* answer. For example:

`? :-parent(fred, mary)` answer: yes (applying clauses 6,3)
`? :-parent(x,y)` answer: 7 pairs (applying clauses 6-12, 2-3)

`? :-mother(sue,x), father(george,x)` - query with more than one atom
answer: james (applying clauses 9,12,7)

`? :- x:father(x,y)` “find all those persons who are fathers”
 Placing “x:” before “father(x,y)” will now return only the value for x, not the corresponding y
answer: x=fred, x=george, x=john
 (applying clauses 6-8)

`? :- grandparent(john,x)` answer: x=mary (applying clauses 6,8,3,1)

- When answering a query, what the PROLOG interpreter does is to prove a query using the program clauses. In theorem proving terminology, each step in the query answering process is a *deduction* using the *resolution* inference rule. The major part of resolution is a pattern matching process called *unification*. During unification, variables become bound to constants and other variables. This is how an answer gets constructed.