

Deductive Data Model

Table of Contents

1	Introduction to Deductive Data Model.....	2
1.1	Data Models	2
1.2	Facts	3
1.3	Variables.....	5
1.4	Logical Functions	6
1.5	Rules of Inference	8
1.6	Goals.....	11
1.7	Architecture of Deductive Database Systems	14
1.8	Backward Chaining Procedure.....	16
1.9	Forward Chaining Procedure.....	18
1.10	Discussion	19
1.11	Recursive rules of inference.....	20
1.12	Computational terms	22
1.13	Active rules of inference	23

1 Introduction to Deductive Data Model

1.1 Data Models

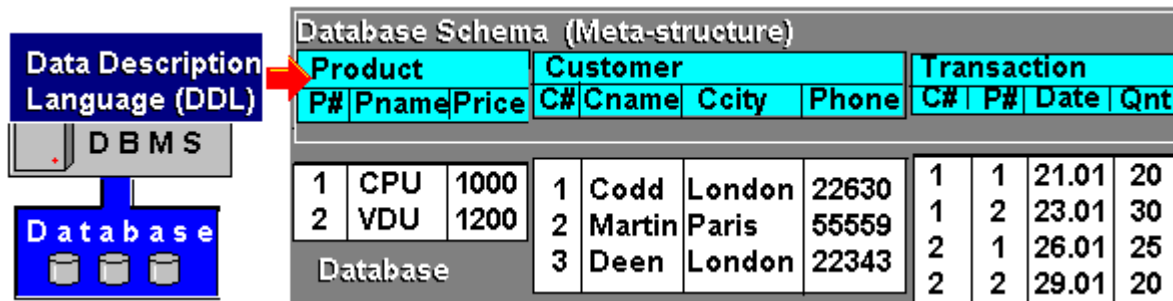
Conceptually, database management is based on the idea of separating a database structure from its contents.

Quite simply, a database structure is a collection of static descriptions of entity classes and relationships between them.

It is perhaps simplest to think of an entity class description as a collection of column headers in a table.

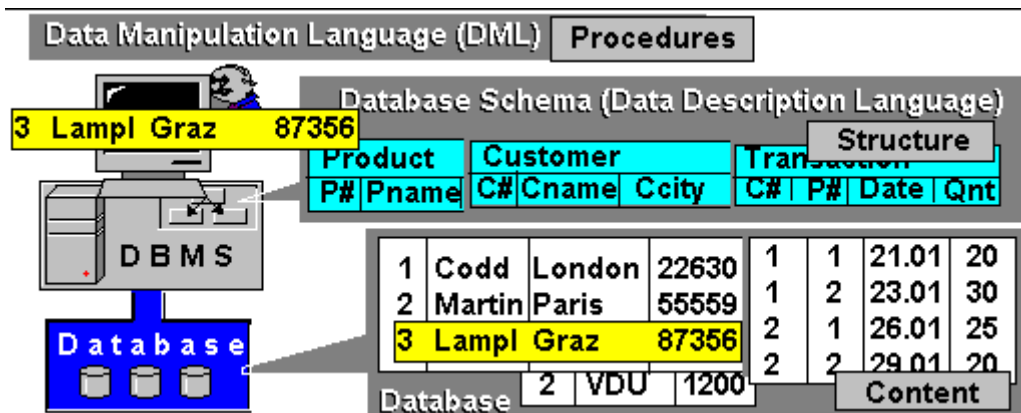
Entity contents can then be thought of as the values that get associated with such headers, creating data objects.

Database schemas are defined by means of a **Data Description Language (DDL)**.



There exist two features of conventional Data Models which are important for a further discussion:

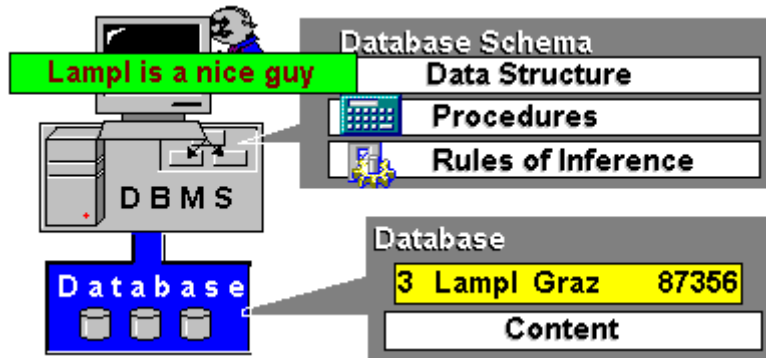
- Data manipulation procedures are separated from the data as such.
- A particular Data Object needs to be explicitly stored into a database to be retrieved afterwards.



Thus, there are two important features which are an inherent of the Deductive Data Model:

Deductive Data Model

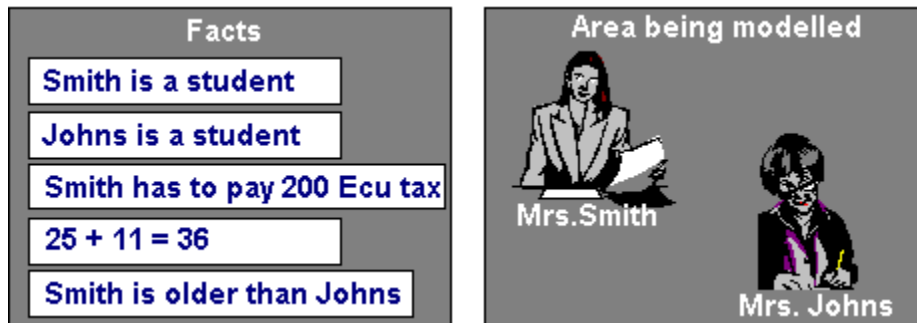
- Deductive Data Model provides an unified approach to definition of Data Structures and Procedures.
- Deductive Data Model provides a possibility to retrieve not only explicitly stored data but **logically inferred data** as well.
- In other words, deductive database systems are able to make logical conclusions.



1.2 Facts

A Logical Data Model operates with so-called **facts**.

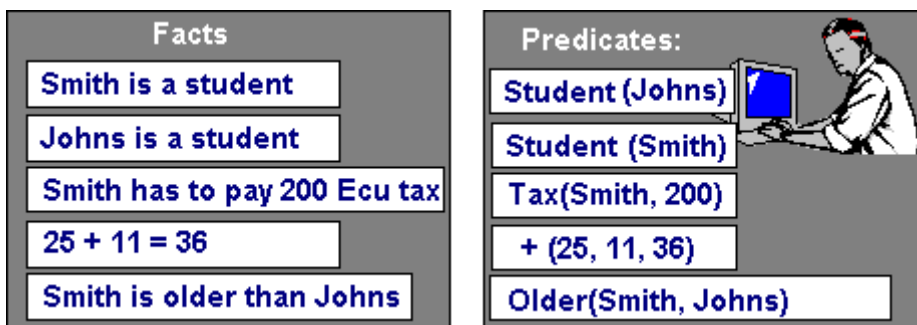
Generally speaking, **fact** is an expression which can be interpreted as **True** or **False**.



In order to be processed by computers, facts need to be coded using a formal notation.

Deductive Database systems operates with facts formally presented as so-called predicates:

Predicate Symbol(Argument, . . .)



Deductive Data Model

Deductive database itself is a collection of so-called **basic facts** (i.e. facts which are known to be "True").

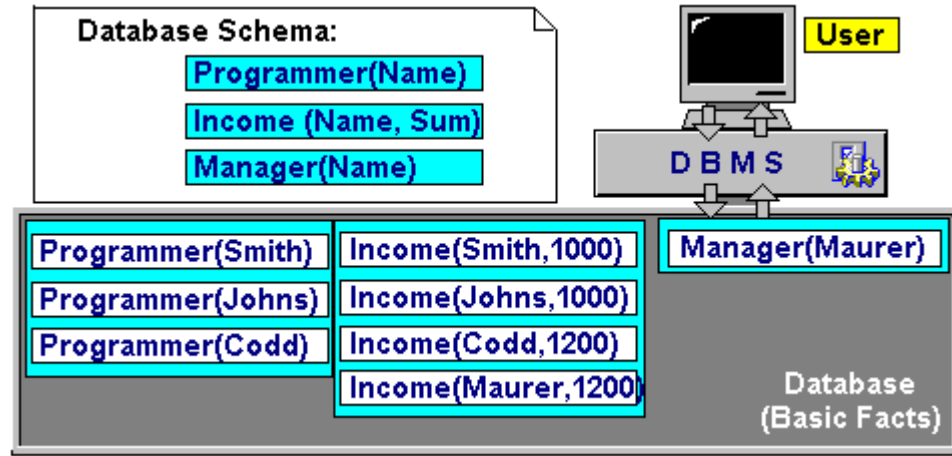
Note, there exist different facts which are represented using the same predicate symbol.

Fact template is a notation to represent a number of similar facts in the following form:

Predicate symbol (Variable, . . .)

Actually, a fact template is a logical function which is valid (i.e. returns the value "True") if the variable values can be found in the database. The function returns "False" otherwise.

Collection of fact templates sufficient to represent any basic fact can be seen as a primitive **deductive database schema**.



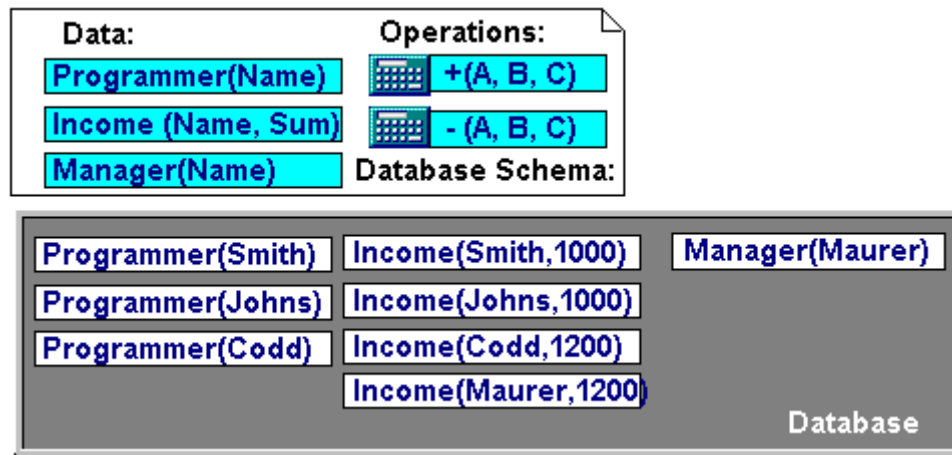
Thus, in deductive databases the information is structured in the form of predicates.

It is important to note that functionality of arithmetical operations and procedures may be presented exactly in the same predicate form.

Consider for example, the arithmetical operation " $A=B+C$ " which can be seen as a logical function (i.e. fact template): $+(A, B, C)$;

- The fact " $+(25, 14, 11)$ " is true because $25 = 14 + 11$.
- The fact " $+(27, 8, 16)$ " is false because $24 = 8 + 16$.

Thus, a deductive database schema may incorporate arithmetical operations presented as fact templates.



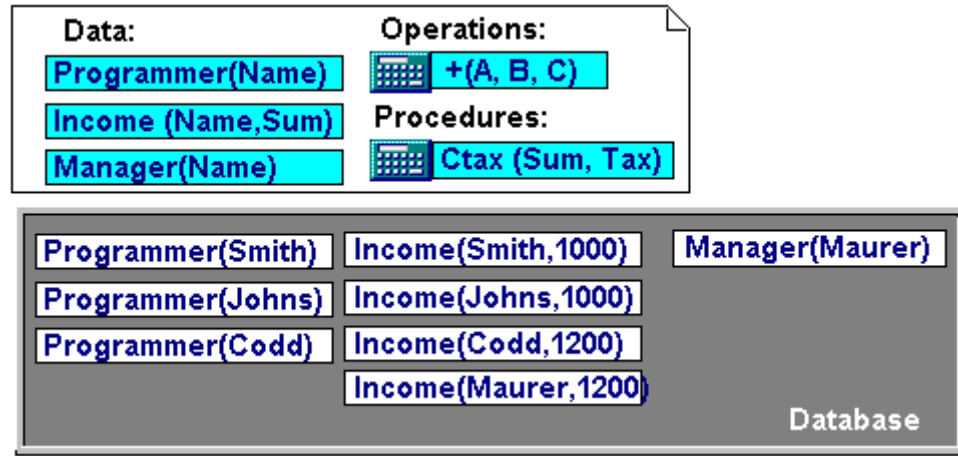
Deductive Data Model

In a similar way, any procedure calculating output values and accepting input values may be seen as a predicate.

Consider for example, the procedure "**Ctax (Sum, Tax)**". Suppose the procedure accepts an input value "**Sum**" - **Total Income** and calculates an output value "**Tax**" - **tax to be paid**.

The procedure can be seen as a logical function (i.e. fact template): "**Ctax (Sum, Tax)**"; The fact "**Ctax (1000, 450)**" is true if the function returns 450 having 1000 as an input and is false otherwise.

Thus, a deductive database schema may incorporate arbitrary procedures presented as fact templates.

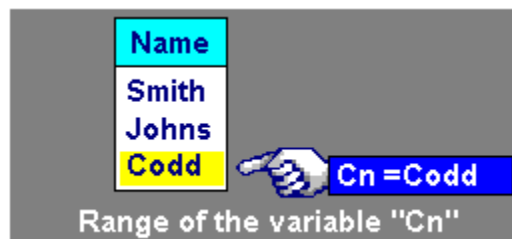


1.3 Variables

Variables are special identifiers to hold a data item during data processing.

For instance, we may interpret the name "**Cn**" as a variable which ranges over **Person Names**.

Thus, the variable "**Cn**" holds one particular **Person Name** at a time.



Similarly, we may also use variables ranging over other domains, or define two and more domain variables which range over data items of one domain.



1.4 Logical Functions

Notation:

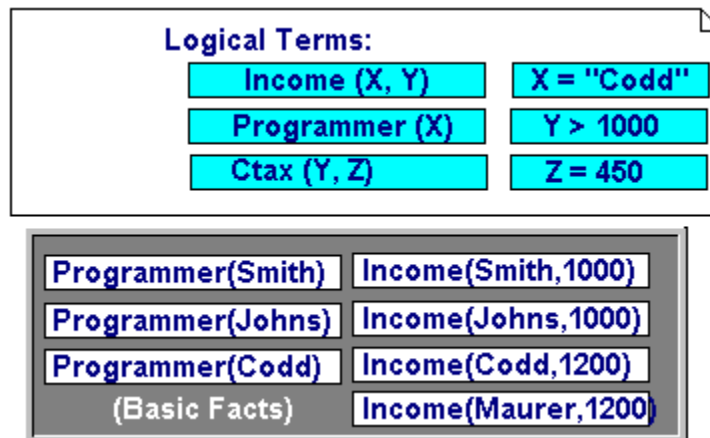
Predicate symbol (Variable, . . .) defines a logical function which returns True or False values depending on a current values of the variables.

Variables (**Variable, . . .**) are called **free variables**.

Note the function **Income(X,Y)** is defined on two **free variables** - "**X**" and "**Y**"

Such primitive logical functions consisting of a **Predicate symbol** and a number of free variables are called **Logical Terms**.

Simple Comparisons which involve variables are also Logical Terms. Note that all variables in comparisons are also **free variables**.

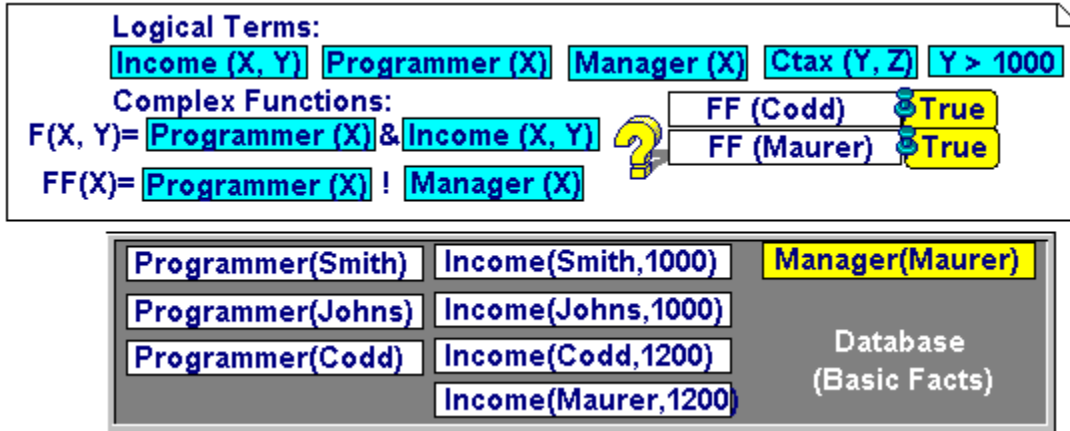


Logical Terms can be can be combined into a new, more complex logical function by means of well-known Logical Operations:

- **Conjunction** (logical "And" denoted as "&"),
- **Disjunction** (logical "Or" denoted as "!") and
- **Negation** (logical "Not" denoted as "~").

Note that the resultant function depends on all free variables defined for the logical terms.

Deductive Data Model



A definition of a logical functions may also include so-called quantifiers. The symbol " \exists " is called an existential quantifier, and can be read as "**there exist at least value of the variable such that ...**".

Informally, the formula " $\exists x F(x)$ " asserts that there exists **at least one** value of **X** (from among its range of values) such that **F(X)** is true.

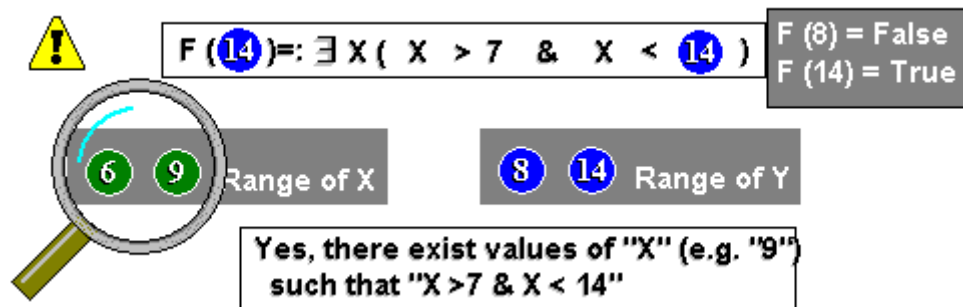
This assertion is false only when **no value of x** can be found to satisfy **F(x)**.

On the other hand, if the assertion is true, there may be more than one such value of X, but **we don't care which**. In other words, the truth of an existentially quantified expression is not a function of the quantified variable(s).

For example,

- the function $F(X,Y)=(X > 7 \& X < Y)$ includes two free variables (X, Y).
- the function $F(Y)=\exists X(X > 7 \& X < Y)$ includes only one free variable (Y).

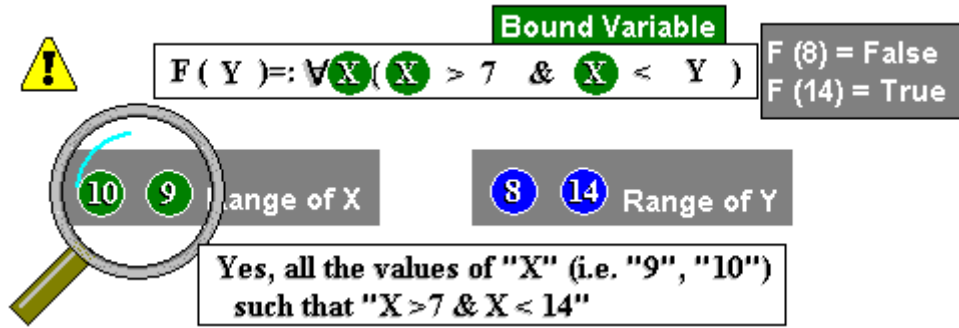
In fact, the value which is produced by this formula does not depend on the current value of the variable x. However, the value does depend on the range of the variable "X". We say, this variable (say, "X") is **bound** by the existential quantifier, or it is a **bound variable**.



Let's turn now to the universal quantifier. The symbol " \forall " is called a universal quantifier and can be read as "**all values such that ...**".

Informally, the formula " $\forall X F(X)$ " asserts that **for every value of X** (from among its range of values) **F(X)** is true.

Like the existential quantifier, the truth of an existentially quantified expression is not a function of the quantified variable(s). In a sense, like the existentially quantified variable, we don't care what the values of X are, as long as every one of them makes the expression true.



Generally, Logical Function is a Well-Formed Formula (WFF) defined by the following rules:

1. $P(A \dots)$ is a WFF if P - predicate symbol, $A \dots$ - free variables
For instance, Programmer (x), Salary (a, b)
2. $A \theta M$ is a WFF if A - variable, M - constant/variable and $\theta \in \{=, >, <, \geq, \leq\}$ For instance, $x = \text{"Smith"}$, $a > b$, ...
3. $F1 \ \& \ F2$ and $F1 \ ! \ F2$ are WFFs if $F1$ and $F2$ are WFFs.
For instance, Programmer(x) & x="Smith" ! Salary (x, y) & y>1200
4. (F) is a WFF if F is a WFF. E.g. Manager(x) & (x="Smith" ! x="Lamp") compare to : Manager(x) & x="Smith" ! x="Lamp"
5. $\exists x (F(x))$ - are WFFs, if $F(x)$ is a WFF having a free variable x .
 $\forall x (F(x))$ The variable x is called a Bound Variable.

1.5 Rules of Inference

Rules of inference are defined as: **Premice(Body) \rightarrow Conclusion(Head)** where
Conclusion(Head) is a term Symbol(x_1, x_2, \dots)

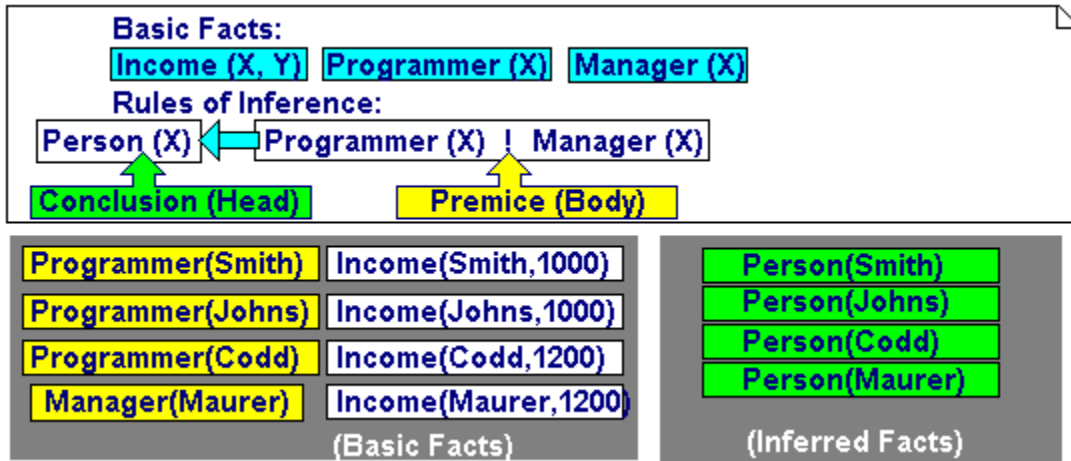
Premice(Body) is a logical function having (x_1, x_2, \dots) as free variables.

The rules of inference are interpreted as follows:

"if for a particular combination of values of the free variables the Body is valid (True) then the Conclusion is also valid (True) for the same values of free variables".

Thus, rules of inference define new, **inferred facts**.

Deductive Data Model



Consider another example:

The Facts "**Prog**" define all employees having a programming experience.

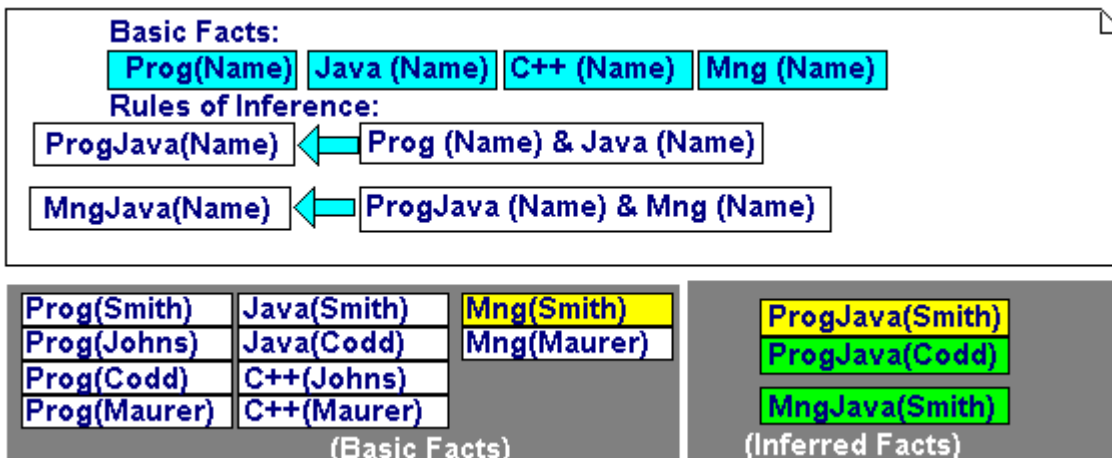
The Facts "**Java**" and "**C++**" define such employees which know a corresponding programming language.

The Facts "**Mng**" define all employees experienced in a project management.

Suppose we need to infer new fact "**ProgJava(Name)**" which shows all the employees having a programming experience with Java.

It should be especially noted that the inferred facts "**ProgJava(Name)**" may be used as logical terms to infer new facts.

Suppose we need to infer a new fact "**MngJava(Name)**" which shows all the employees which can manage a Java software development.



Consider another example:

The Facts "**Offer(Shop,Product)**" define Shops and Products which are offered by the Shops.

The Facts "**Need(Name,Product)**" define Persons (Name) and Products which are currently needed.

Suppose we need to infer new fact "**Visit(Name, Shop)**" which shows which Shop should be visited by a particular person (Name). On the first glance we could define it as:

Offer(Shop,Product) & Need(Name,Product) → Visit(Name,Shop)

Note, the rule of inference above is ambiguous. Thus, for example, we cannot come to a

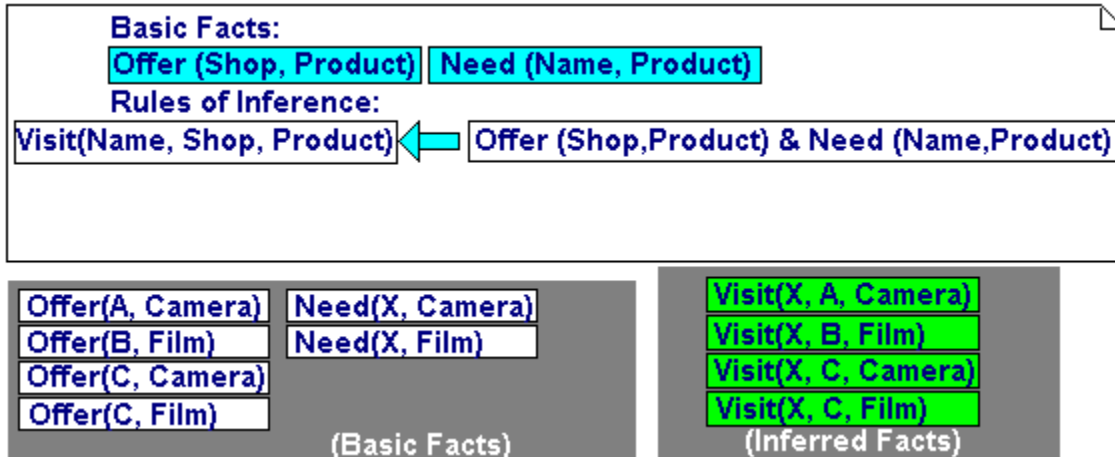
Deductive Data Model

particular conclusion whether the fact **Visit (X, A)** is valid or not.

The Premise "**Offer ("A", Product) & Need ("X", Product)**" is valid if **Product="Camera"**, and it is not valid if **Product="Film"**.

Formally, head of a rule of inference always depends on the same free variables as the corresponding body. Note that the rule above violates this precondition.

A simplest way to avoid the ambiguity is to include the third variable **Product** into the head.



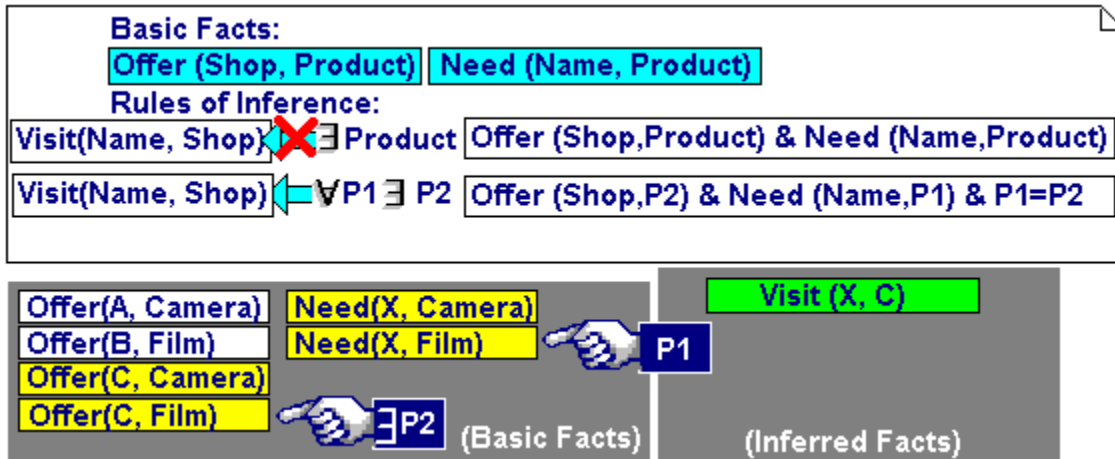
Of course, there is another, much more elegant way around to eliminate the ambiguity. Thus, an **Existential Quantifier** can be used to bind the variable Product.

The following facts are inferred using this rule of inference:

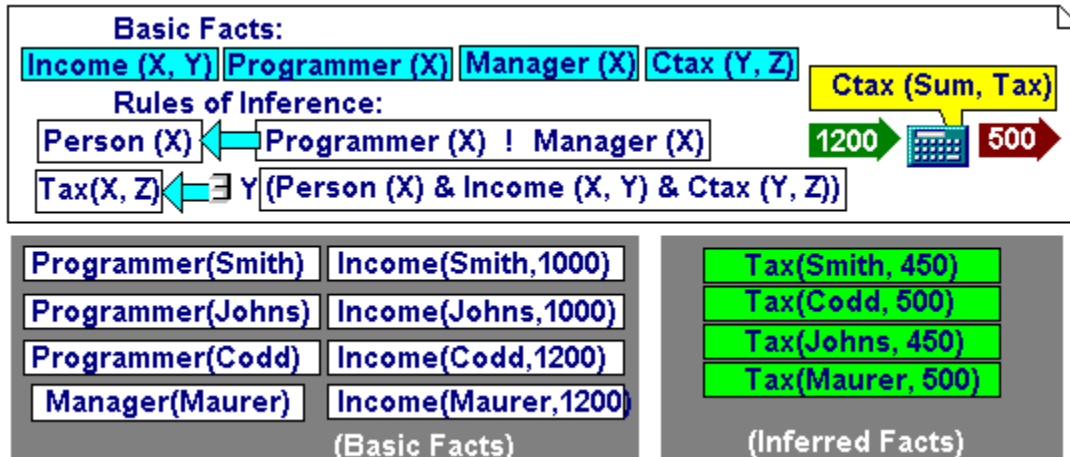


Note that the rule of inference is not "intelligent" enough to infer a shop name offering all the products which are needed.

We need to apply an **Universal Quantifier** to define such an "intelligent" rule of inference:



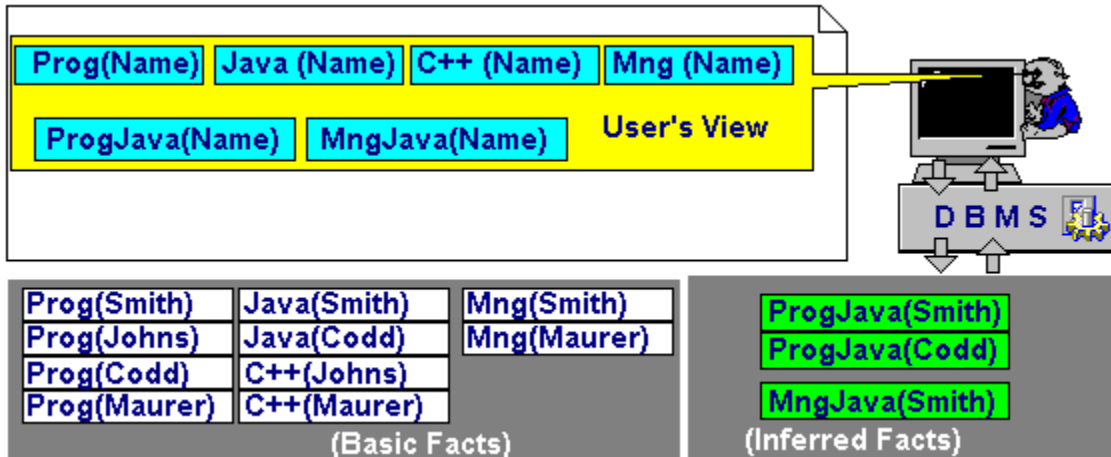
External procedures are embedded into rules of inference as ordinary logical terms. New facts are inferred by means of **automatical calculation** of variables by means of the predefined procedure.



1.6 Goals

Thus, a deductive database consist of basic facts. Templates for all such basic facts are described in the current database schema. Additionally, the database schema defines so-called rules of inference which are used to automatically infer new facts (so-called inferred facts). Inferred facts become available to the users in the same way (i.e. through an unified interface) as basic facts.

Deductive Data Model



Users communicate with a deductive database by means of so-called **Goals**.

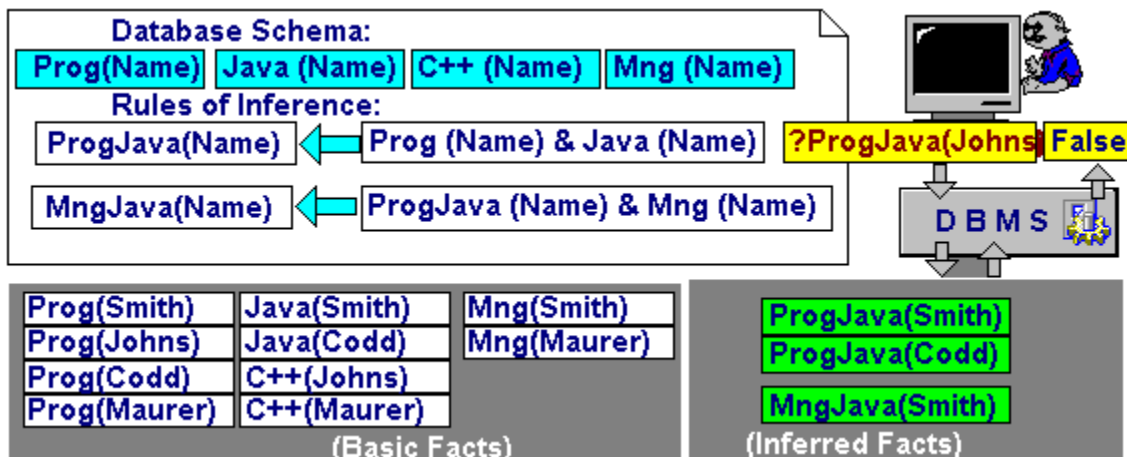
There are so-called **Verifier Goals**, **Finder Goals** and **Doer Goals**.

Verifier Goal is a predicate which can be True or False.

For example, **? Prog(Smith)**

Thus, users may simply input such verifier goals to a deductive database system to receive "True" or "False" as a result. A dialog between a user and deductive DBMS could look as follows:

- **User: ? Prog(Smith)**
- **DBMS: True**
- **User: ? Prog(Lampl)**
- **DBMS: False**
- **User: ? ProgJava(Smith)**
- **DBMS: True**
- **User: ? ProgJava(Johns)**
- **DBMS: False**



Deductive Data Model

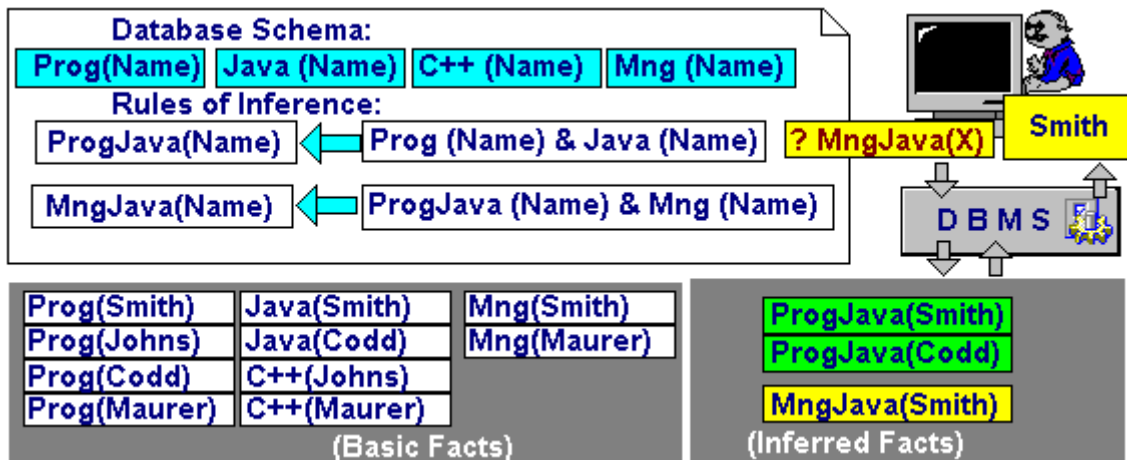
Finder Goal is a logical function defined with at least one free variable.

For example: $? \text{Java}(x)$

In other words, users simply input finder goals to find such values of the free variable for which the goal is valid (i.e. the function is True).

Thus, a dialog between a user and deductive DBMS could look as follows:

- **User: ? Prog(Programmers)**
- **DBMS: Programmers:Smith, Johns, Codd, Maurer**
- **User: ? MngJava(Candidates)**
- **DBMS: Candidates: Maurer**

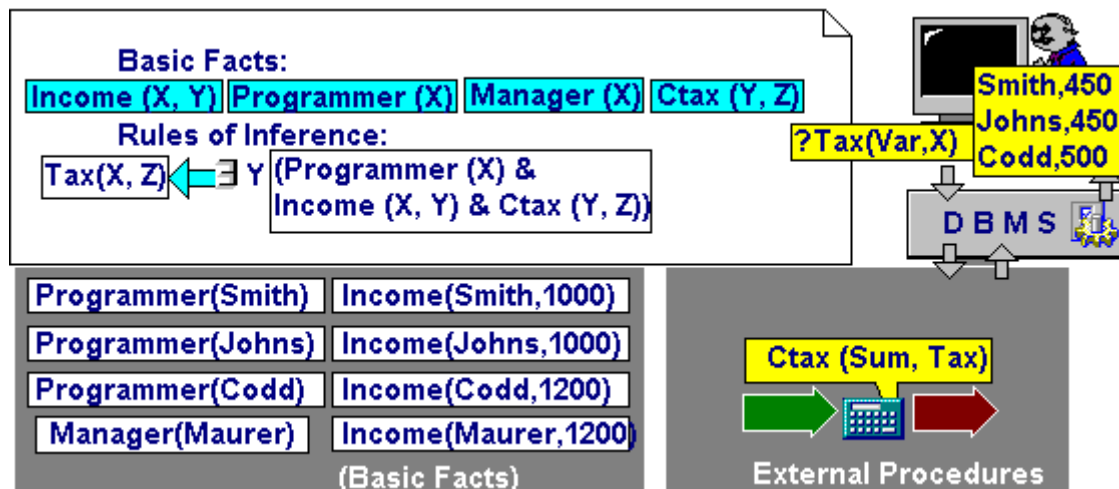


Finder Goal can be applied to facts inferred by means of rules of inference referring to external procedures.

For example: $? \text{Tax}(\text{Smith}, X)$

Deductive DBMS invokes the external procedure dynamically to calculate the resultant values.

Note, in this particular case the Finder Goal may be defined with two variables as well:



Doer Goal is an updating operation which is applied to a database (i.e. to basic facts). For

Deductive Data Model

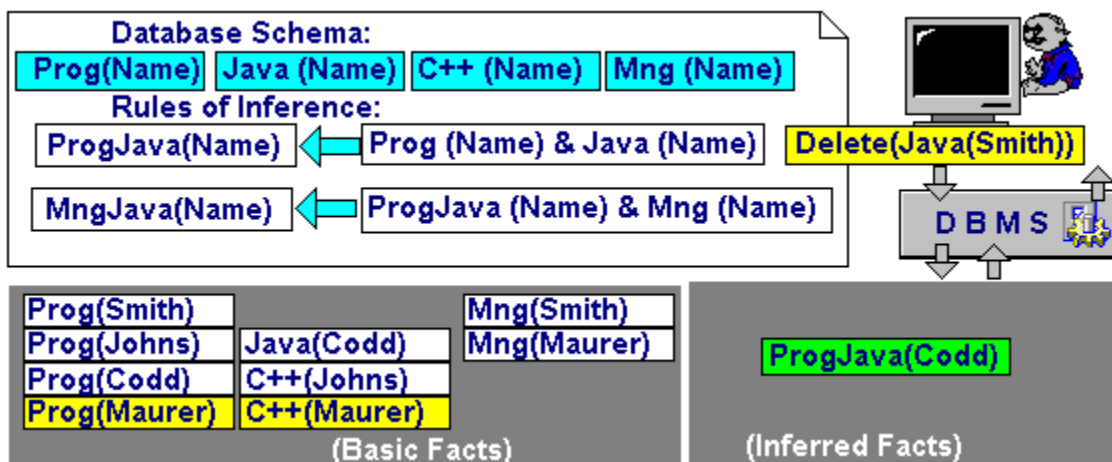
example, ? Add(Prog(Maurer))

Thus, users simply input such doer goals to update a current database of basic facts.

Note that inferred facts are generally affected by such doer goals.

Thus, a dialog between a user and deductive DBMS could look as follows:

- **User: ? Prog(Programmers)**
- **DBMS: Programmers:Smith, Johns, Codd**
- **User: ? ADD(Prog(Maurer))**
- **DBMS: OK**
- **User: ? Prog(Programmers)**
- **DBMS: Programmers:Smith, Johns, Codd, Maurer**



1.7 Architecture of Deductive Database Systems

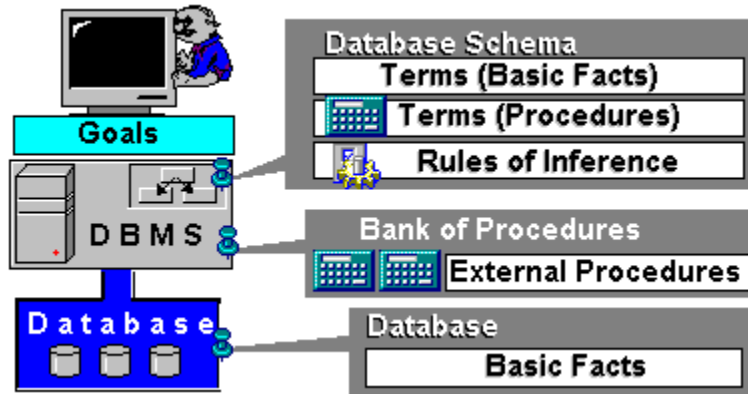
Thus, a deductive database system consists of the following components:

- Basic Facts (Database)
- Procedures which are used to calculate new values (Bank of Procedures)
- Non-procedural description of a system functionality (Rules of Inference)

All components (i.e. basic facts, procedures and inferred facts are represented as predicates, hence, the users are provided with an unified view to all such components of a deductive database system

Users communicate with a deductive database system by means of so-called verifier, finder and doer goals.

Deductive Data Model



Suppose we need a deductive database to manage a stock of products which are sold via the Internet (i.e. we accept sales orders via the Internet and distribute the products to customers). In this case Basic Facts kept in the database may look as follows:

- **Stock(Prd, Qnt)** Name(Prd) and quantity(Qnt) of a certain product available from a stock.
For example: Stock("Film",100).
- **Supply(Firm, Prd)** title of a firm (Firm) which provides us with a particular product(Prd).
For example: Supply("Codak", "Film").
- **Req(Prd, Qnt)** actual request for a product (Prd) in a quantity (Qnt).
For example: Req("Film",10).

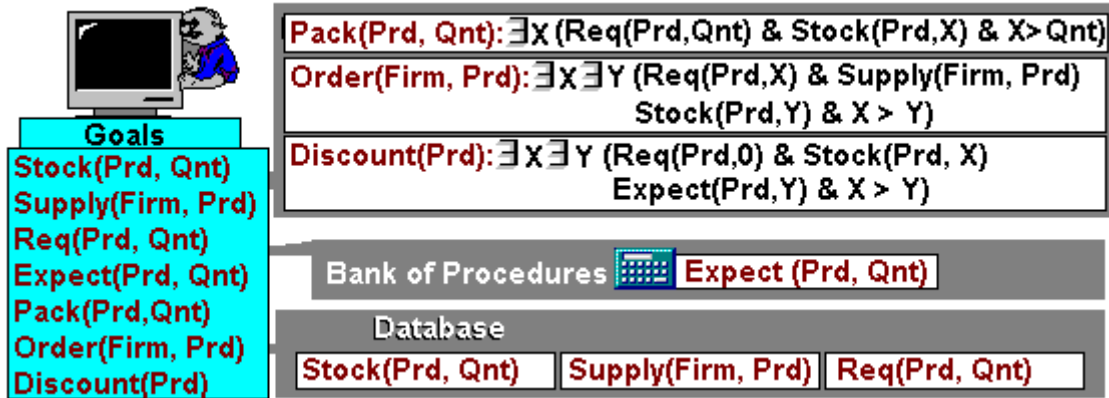
Suppose, there also is a procedure **Expect(Prd, Qnt)** which calculates somehow an assessment of expected quantity(Qnt) of product(Prd) which will be required soon.

For example: Expect("Film", 200).

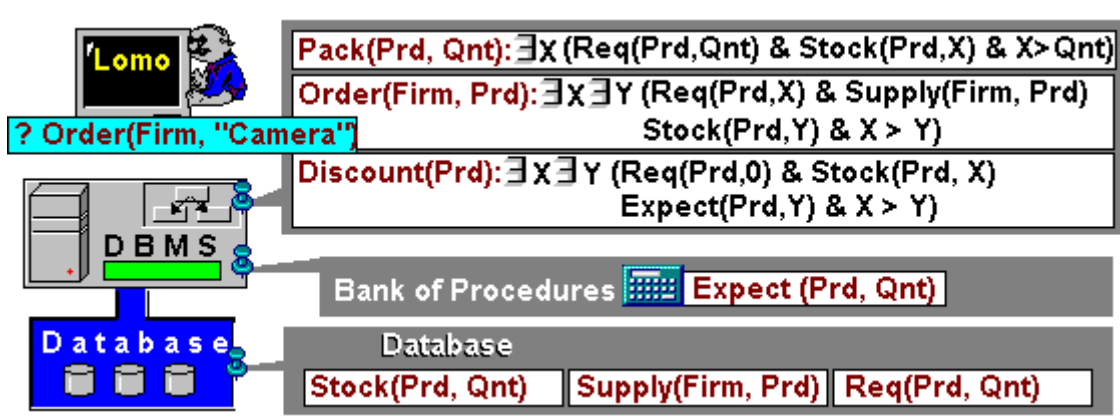
The following rules of inference might be defined:

- **Pack(Prd, Qnt)** the inferred fact is true if Qnt pieces of a product(Prd) should be packed and send to customers.
- **Order(Firm, Prd)** the fact is true if we should order a product(Prd) to a corresponding firm (Firm).
- **Discount(Prd)** the fact is true if a product(Prd) should be discounted to increase a customers demand.

Users can view this deductive database system as the following collection of predicates.



As we can see the system demonstrates a sort of "intelligent" behavior. The users define their questions to the system in a form of goals and the system answers:

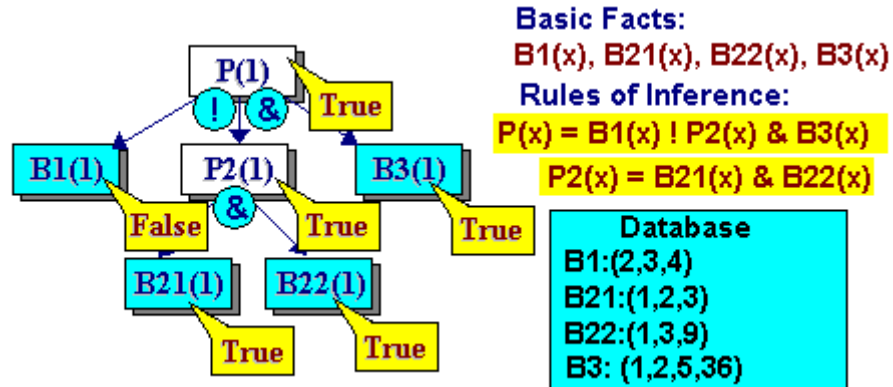


1.8 Backward Chaining Procedure

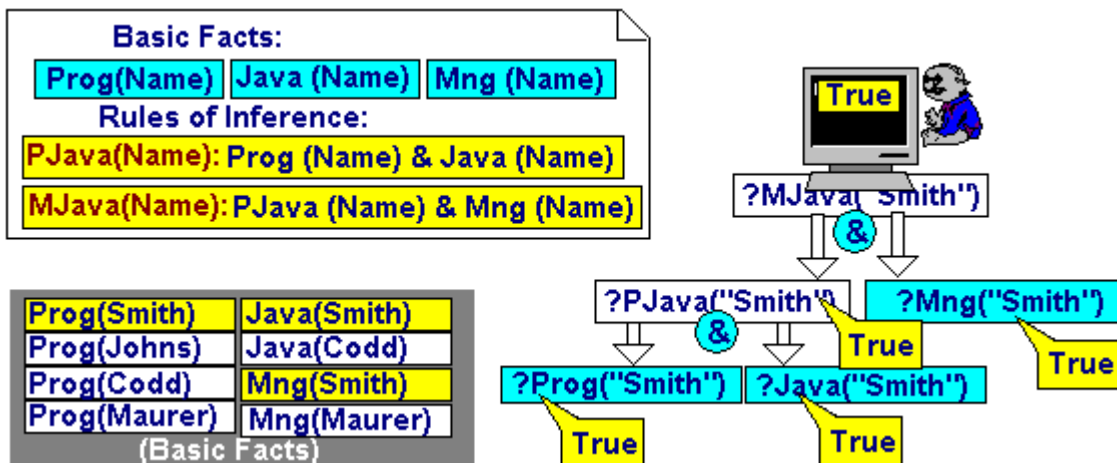
How do deductive DBMS infer a response to a given goal?

- Phase 1: A given goal is reduced to simple subgoals corresponding to basic facts or procedures.
- Phase 2: Searching for values for the simple subgoals and putting them together to form the response.

Deductive Data Model



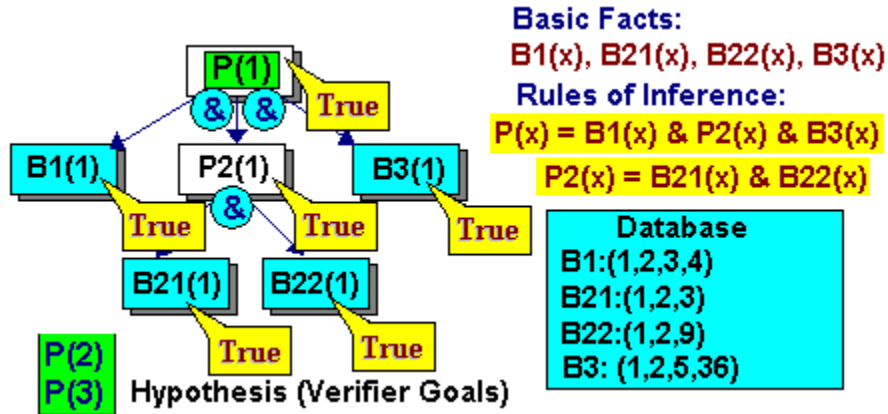
Consider the previously discussed database dealing with information on programmers and managers employed by a software firm.
 A Verifier Goal is interpreted as follows:



Finder goals are interpreted in a similar way:

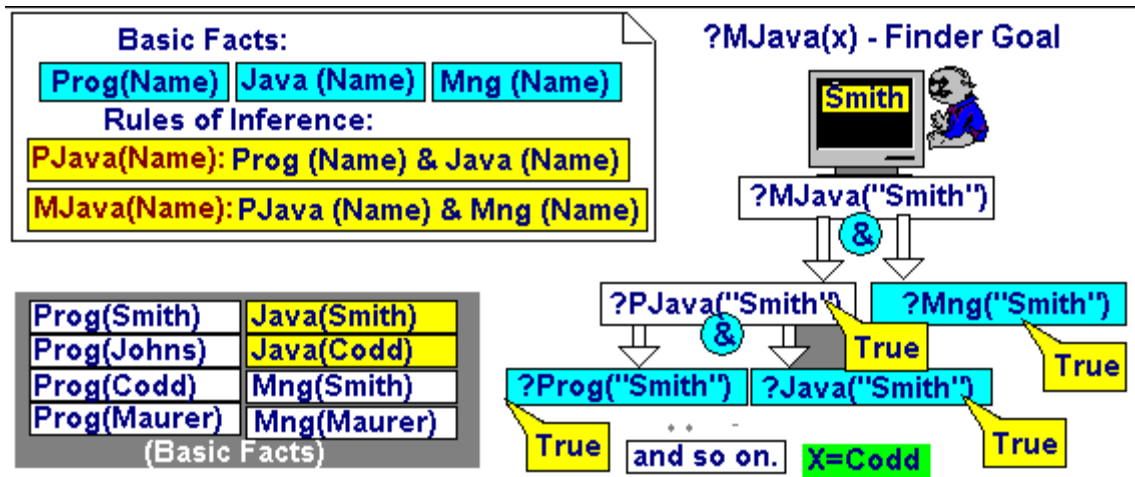
- Phase 1: A given goal is reduced to simple subgoals corresponding to basic facts or procedures.
- Phase 2: A particularly selected term is used to select a number of basic facts which define all potentially possible values of the free variables. Such selected values are called **hypothesis**.
- Phase 3: All **hypothesis** are checked out as Verifier Goals to select valid values of free variables.

Deductive Data Model



Consider once again the previously discussed database dealing with information on programmers and managers employed by a software firm.

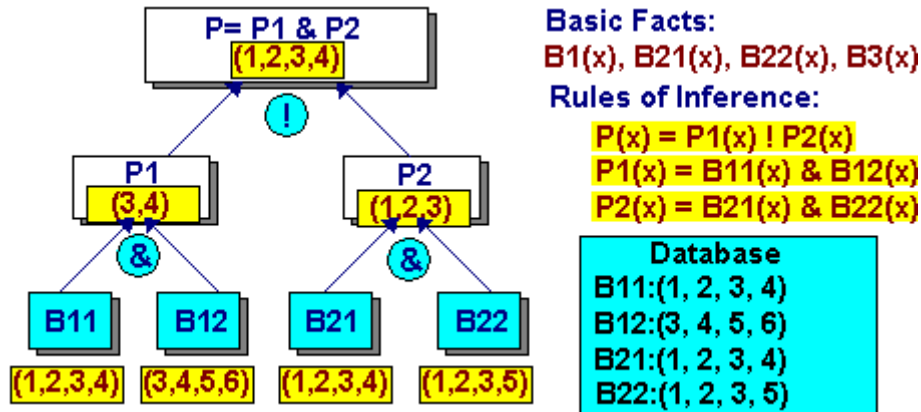
A Finder Goal is interpreted as follows:



1.9 Forward Chaining Procedure

Counterpart for the Backward Chaining Procedure is called **Forward Chaining Procedure**. The Forward Chaining Procedure deals with putting basic facts together to infer **all possible new facts**.

Deductive Data Model



Consider the same database as before: The **Forward Chaining Procedure** can be applied to infer all facts defined by the rules of inference:



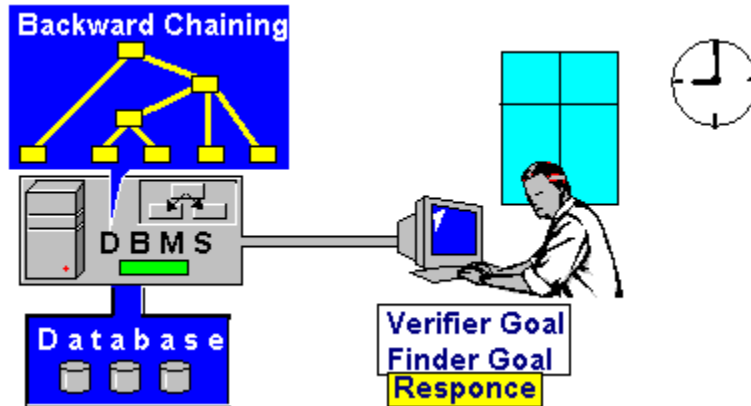
1.10 Discussion

Note that the **Backward Chaining Procedure** needs to be invoked every time when the user issues a **Verifier** or **Finder Goal**.

The response is inferred dynamically and NO inferred data are stored into the database.

Since, interpreting a Finder goal may take considerable time, the **Backward Chaining Procedure** is normally applied for database systems which are not critical to response time.

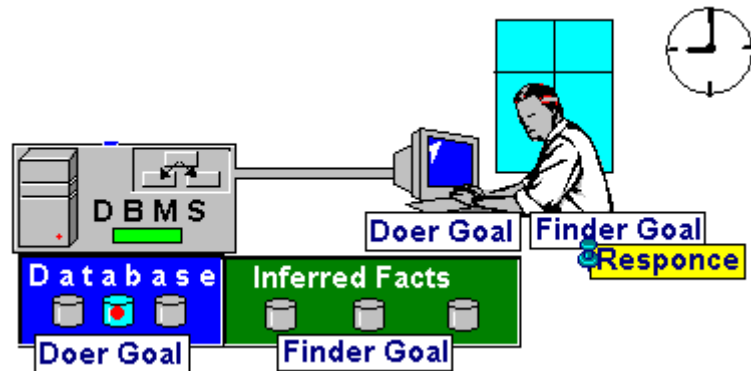
Deductive Data Model



On the contrary, the **Forward Chaining Procedure** needs to be invoked only when the database is modified (i.e. when the user issues a **Doer Goal**).

All facts defined by means of rules of inference are inferred and stored into the database. Thus, forthcoming **Verifier** or **Finder Goals** can be interpreted by means of searching such previously stored inferred fact.

Deductive systems which are supposed to control processes in real time (say, an oil refinery process), normally supports **Forward Chaining Procedure** to minimize a response time.



1.11 Recursive rules of inference

It should be especially noted that inferred facts can be defined recursively, i.e. an inferred fact can be used as a **Conclusion** and as a predicate in the **Body** of a particular rule of inference.

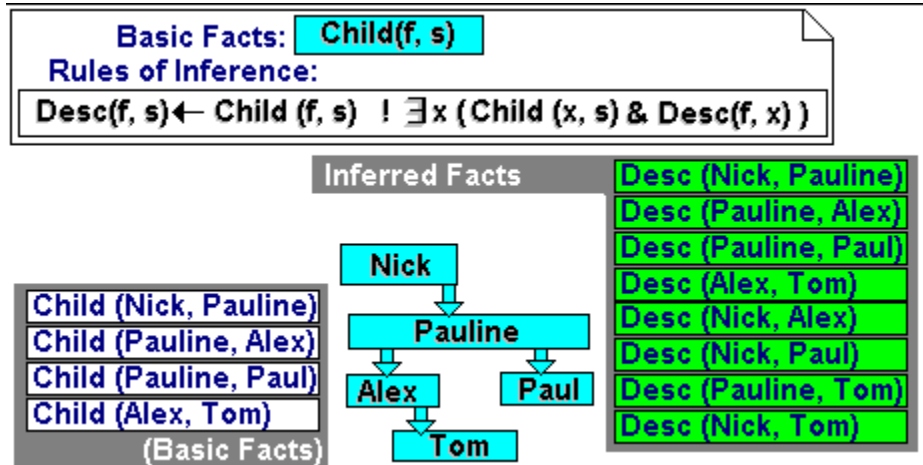
Consider the following database schema:

- **Child(f, s)** basic fact: "person "s" is a child of person "f" .

Rule of inference: **Desc(f, s)**: Person "s" is a descendent of person "f" if

- person "s" is a child of person "f" or
- there exists such person "x" that this person "x" is a descendant of "f" and the person "s" is a child of "x".

The facts **Desc(f, s)** are inferred as follows:

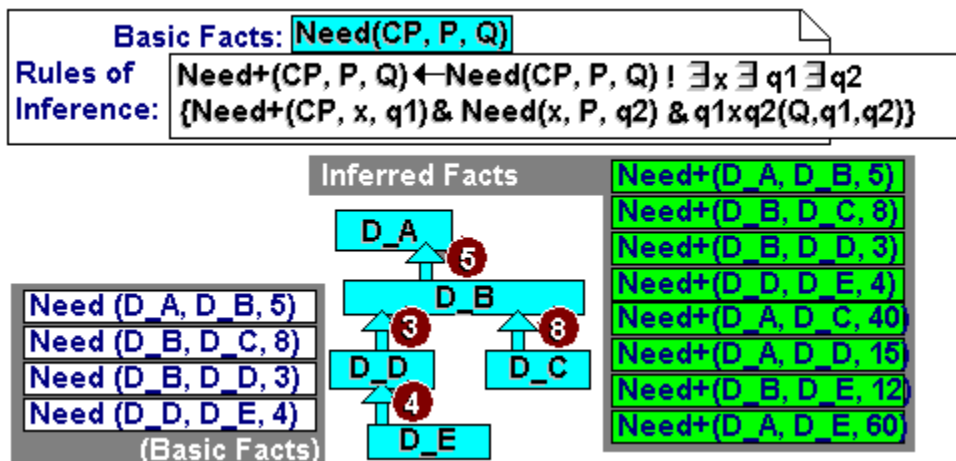


Recursive rules of inference may invoke external procedures or arithmetical operation (for example, multiplication " $Q=q1 \times q2$ ") defined as predicates.

Consider another example of a deductive database system:

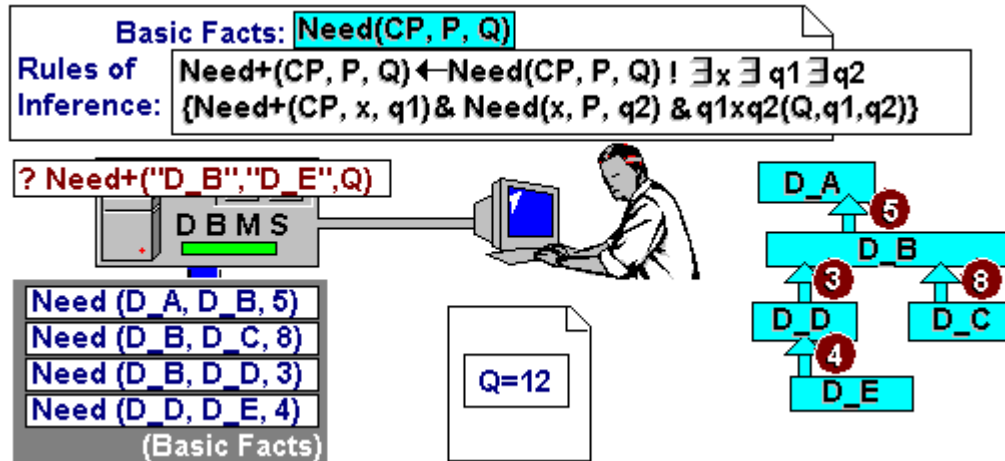
- **Need(CP, P, Q)** basic fact: "a particular quantity ("Q") of the product "P" is needed for a manufacturing operation producing a new product "CP".
- **Need+(CP, P, Q)**: inferred fact: "a particular quantity ("Q") of the product "P" is needed for a manufacturing operation producing a new product "CP". for a manufacturing operation producing a new product "CP" or for manufacturing all necessary components of the product "CP".

The facts **Need+(CP, P, Q)** are inferred as follows:



Recursive rules of inference combined with calculations are a very powerful tool. Thus the deductive system discussed before might be a very useful to support so-called "Bill of Materials". It "intelligently" infer an answer to questions like:

- "print all materials necessary to produce "D_A"" - Finder goal: **?Need+("D_A", Prd, Qnt)**
- "how many pieces of "D_E" are needed to produce "D_B"" - Finder goal: **?Need+("D_B", "D_E", Qnt)**, etc.



1.12 Computational terms

In order to provide rules of inference with a sufficient computational power, the logical Data Model also supports a concept of so-called **computational terms**.

Primitively speaking, computational terms are arithmetical expressions having free variables, constants and/or other terms as parameters.

The following notation is used to define such computational terms:

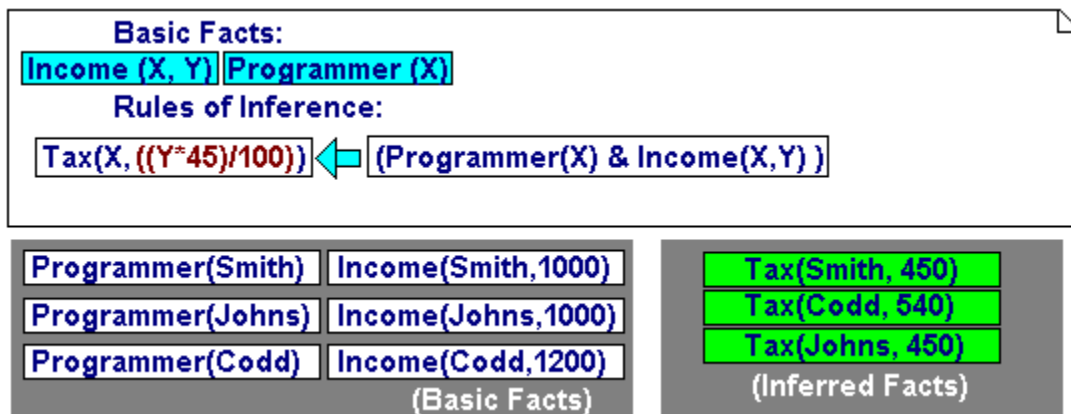
([Parameter_1] [Operation] [Parameter_2])

where [Parameter_1] and [Parameter_2] are free variables, constants or other computational terms.

Terms are used as free variables in rules of inference. Consider the following database:

If a personal tax is calculated as 45 percents of an income, a "Tax" rule of inference might be defined as below.

It should be noted that although computational terms can be replaced with arithmetical operations in a predicate form, the computational terms reasonably simplify the definition of rules of inference.

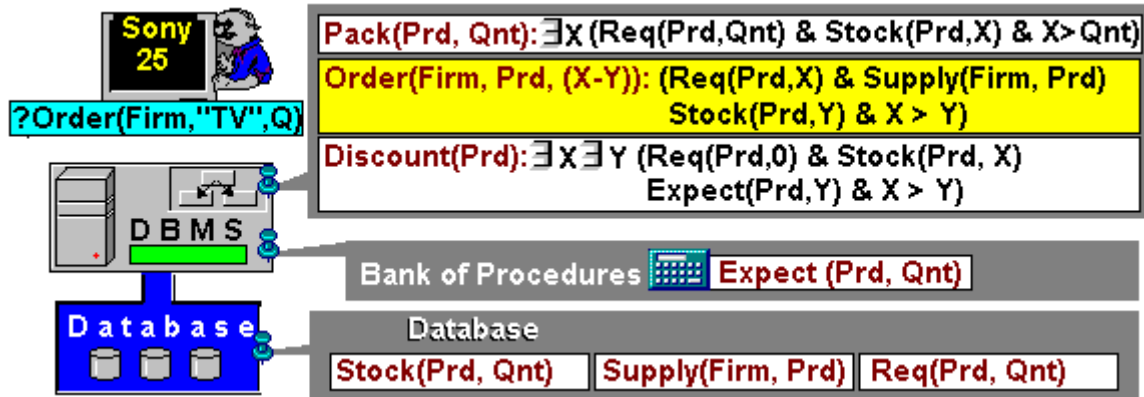


Recollect the previously discussed database schema.

The rule which infers the fact "Order" is too primitive and does not infer any information on how many pieces of the product (Prd) are required.

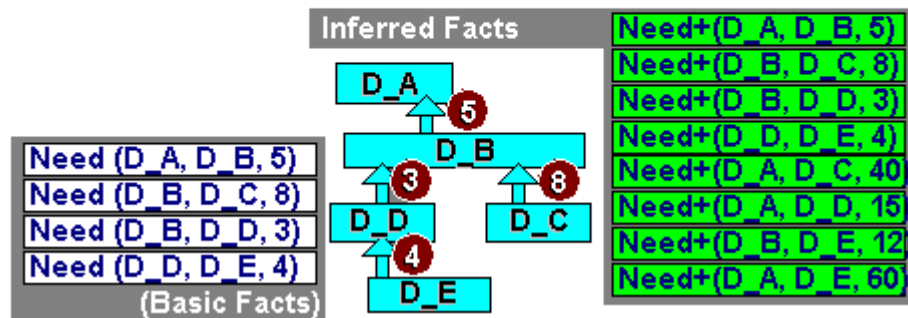
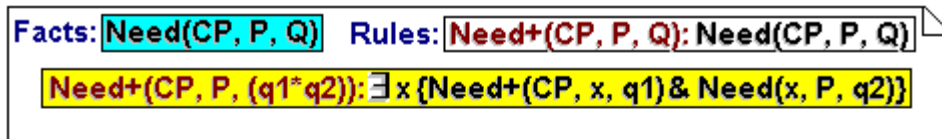
Thus, the system functionality may be considerably improved if the rule is defined using a computational term.

The improved rule of inference demonstrates more "intelligent" behaviour.



Two issues should be mentioned:

- 1. A particular fact may be inferred by two or more rules of inference. In this case, a particular fact is valid if it is inferred by at least one of the rules.
- 2. Computational terms may be also used in recursive rules of inference.



1.13 Active rules of inference

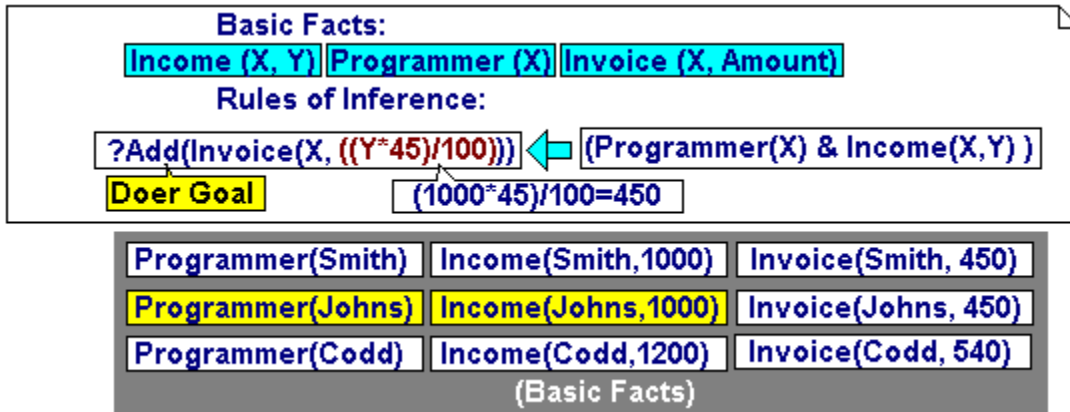
Simply stated, the rules of inference may infer not only new facts but also **Doer Goals** modifying a database therefor.

Such rules of inference are called **Active Rules of Inference**.

They are interpreted as follows:

- If a body of a particular active rule of inference is valid for a combination of free variables, the Doer Goal specified as a head, is evaluated automatically with the same values of free variables as parameters.

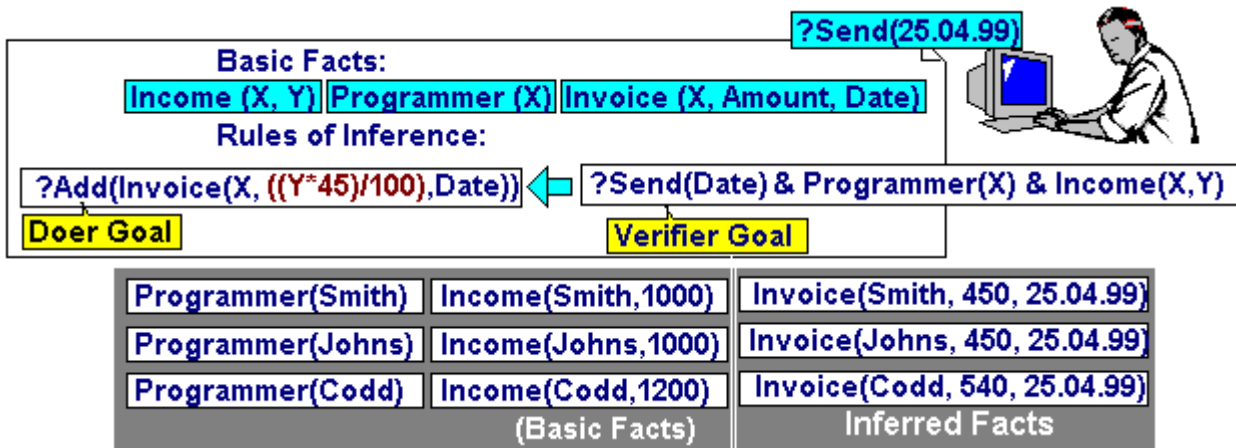
Deductive Data Model



Since Active Rules inference Doer Goals and modify database, a particular moment of time when the rule should be evaluated, plays an important role.

For example, the basic facts "Invoice" should be provided with a particular date and, hence, generated on this particular date.

Normally, Verifier goals are used in definition of active rules of inference as "triggers". Thus, the following active rule of inference is evaluated when an user explicitly issues the verifier goal ?Send("Date").



Thus, if an deductive database system is provided with a number of active rules of inference, users may utilize the system functionality via verifier goals without actual knowledge on the database structure.

For example two active rules below, allow users to issue commands "Generate tax invoices to all programmers", "Programmer X has paid out the tax invoice", etc. via primitive Verifier goals Send("Date"), Receive("X"), etc.

Deductive Data Model