

# Network (CODASYL) Data Model

## Table of Contents

<b>1</b>	<b>Network (CODASYL) Data Model.....</b>	<b>3</b>
1.1	Introduction .....	3
1.2	Database Record.....	4
1.3	Data Base Key .....	6
1.4	Data Set .....	6
<b>2</b>	<b>Bachmann Diagrams &amp; Data Manipulation.....</b>	<b>9</b>
2.1	Introduction .....	9
2.2	Bachmann Diagram .....	9
2.3	Data Updating Facilities .....	11
2.4	Network Subschema .....	13
<b>3</b>	<b>CODASYL D D L.....</b>	<b>16</b>
3.1	Introduction .....	16
3.2	"Record" Clause .....	16
3.3	"Location Mode" Sub-clause.....	17
3.4	Example.....	20
<b>4</b>	<b>CODASYL D D L (Part 2).....</b>	<b>22</b>
4.1	Introduction .....	22
4.2	Set clause.....	23
4.3	Insertion sub-clause .....	25
4.4	Retention sub-clause.....	25
4.5	Order sub-clause.....	26
4.6	Illustrative Example .....	27
<b>5</b>	<b>Data Manipulation Facilities.....</b>	<b>29</b>
5.1	Introduction .....	29
5.2	Application environment.....	29
5.3	Currency indicators .....	30
5.4	Record templates .....	33
5.5	Error Status.....	34
<b>6</b>	<b>C O D A S Y L D M L (Part 1) .....</b>	<b>35</b>
6.1	Introduction .....	35
6.2	Finding a Record Directly .....	36
6.3	Scanning a Set Occurrence.....	37
6.4	Finding an Owner.....	39
<b>7</b>	<b>C O D A S Y L D M L (Part 2) .....</b>	<b>41</b>
7.1	Introduction .....	41
7.2	STORE Operator .....	41

## Network (CODASYL) Data Model

7.3	INSERT Operator.....	42
7.4	REMOVE Operator.....	44
7.5	MODIFY Operator.....	45
7.6	DELETE Operator.....	46

# 1 Network (CODASYL) Data Model

## 1.1 Introduction

There are a variety of different ways to organize data within a data base. A particular method selected for structuring information within a database is called a **Logical Data Model**.

Thus, a Logical Data Model specifies:

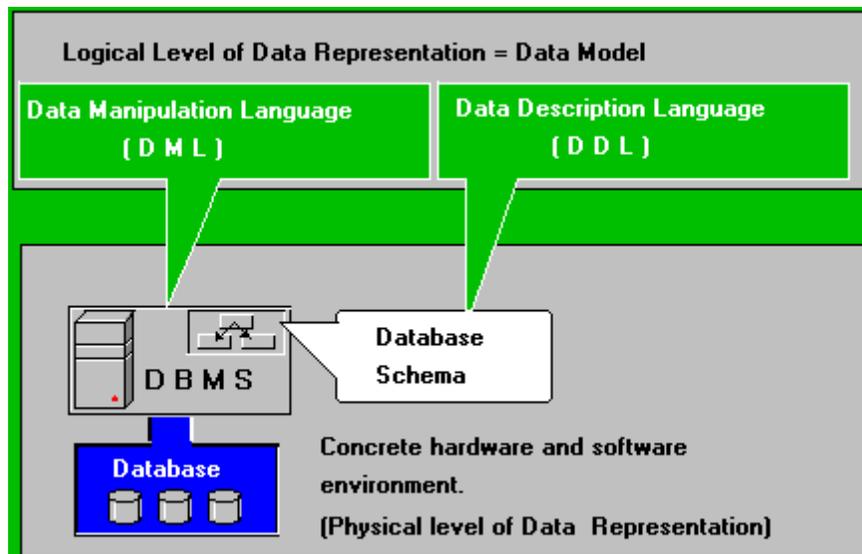
- 1. the rules according to which data are structured;
- 2. the associated operations that are permitted.

It may also be seen as a technique for the formal description of data structure, usage constraints and operations. The facilities available vary from one Logical Data Model to another.

We can say: each DBMS maintains a particular data model.

More formally: A Data Model is a combination of at least three components:

- 1. A collection of data structure types.
- 2. A collection of operators or rules of inference, which can be applied to any valid instance of the data types listed in (1).
- 3. A collection of general integrity rules, which implicitly or explicitly define the set of consistent data base states or change of state or both.



There can be two or more different DBMS which support the same Logical Data Model. Thus, knowledge of at least one Logical Data Model is sufficient to develop Data Base applications. Be careful not to mix the terms "*Information model*" and "*Data model*".

An information model is a description of the "real world" in the terms of (by means of) a Logical Data Model.

**The Network Data Model (NDM)** was proposed by the Data Base Task Group (DBTG) of the Programming Language Committee (subsequently renamed the COBOL committee)

## Network (CODASYL) Data Model

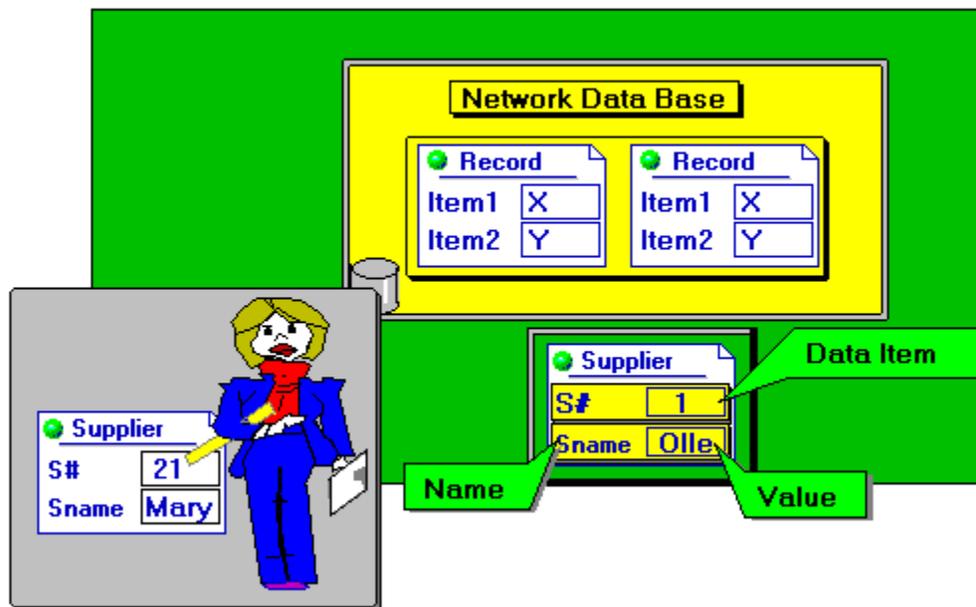
of the "Conference on Data Systems Language" (**CODASYL**), the organisation responsible for the definition of the COBOL programming language.

The Network Data Model is also known as the "**CODASYL Data Model**" or sometimes as the "**DBTG Data Model**". The DBTG final report was produced in 1971. The DBTG report contained proposals for three distinct database languages:

- a schema data description language;
- a subschema data description language;
- a data manipulation language.

### 1.2 Database Record

A network data base consists of so-called records. A Record ( i.e. Record Occurrence ) is a collection of data items.

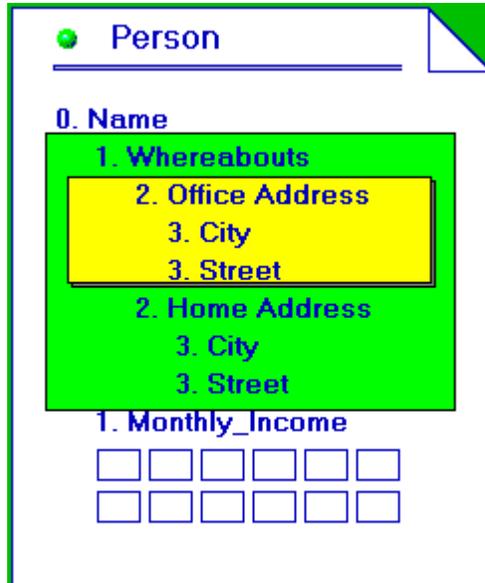


Each data item has a name and a value. Every record describes some real person, object or event of the area being modelled.

A network Database Management System ( DBMS ) operates on records. A record (or, more precisely, record occurrence) is a collection of data items which can be retrieved from a data base, or which can be stored in a data base as an undivided object. Thus, a DBMS may: STORE, DELETE or MODIFY records within a data base. In this way, a number of records within a network database is dynamically changed.

A CODASYL record may have its own internal structure. Two or more contiguous elementary items may be grouped together to form a group item.

## Network (CODASYL) Data Model



A group item may consist not only of elementary items but also of other group items, hence allowing the user to build up a naming structure.

To avoid confusion, the levels in this structure must be numbered downwards from the top.

A CODASYL record may include so-called tables. A table is collection of values grouped under one name of a data item.

The user references the elements in the table using subscripting similar to an array in ordinary programming. For example:

**Persons.Name.Monthly\_Income[0]**

A CODASYL record allows duplicate names of data items.



Suppose the user needs to refer to the attribute Name which is an item of both records. In order to distinguish one from the other, the user must write:

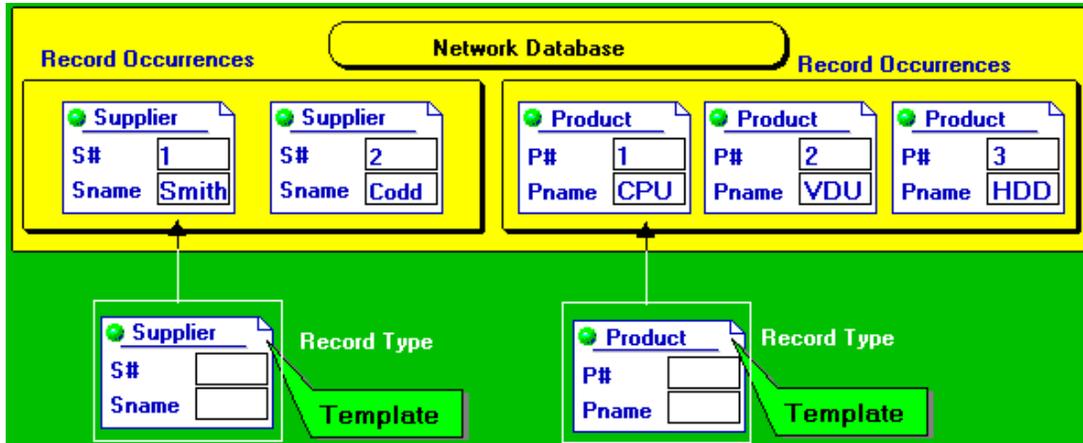
**Supplier.Name** and **Product.Name**

This technique is known as a qualification. On complex case, the qualification carries through all levels of naming within the record. The qualification can be omitted if the user refers to the unique name of a data item.

There may be two or more records with the same internal structure, or more precisely, which include different values of the same attributes.

A collection of such record occurrences is called a **Record Type**. Each record type has a unique name. We can thus say that different record occurrences of a same record type describe different instances of a certain entity of the "real word".

## Network (CODASYL) Data Model



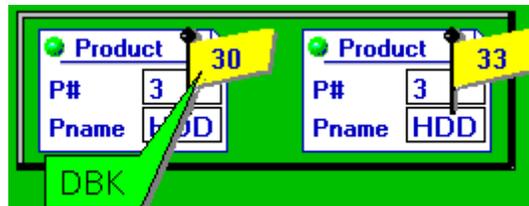
In other words, a record type is a frame (template) for the real data representation. Please remember:

- A record type defines all permissible occurrences.
- All record types must be described in the **Data Base Schema**.

### 1.3 Data Base Key

Two or more different records within a network data base may have duplicate values of all data items.

The **Data Base Key (DBK)** is conceptually a data item, whose value is associated with each stored record in the data base.



We can think of it as a unique internal record identifier used inside a data base to distinguish one record from another. Each record is assigned a data base key value when it is stored in the data base for the first time.

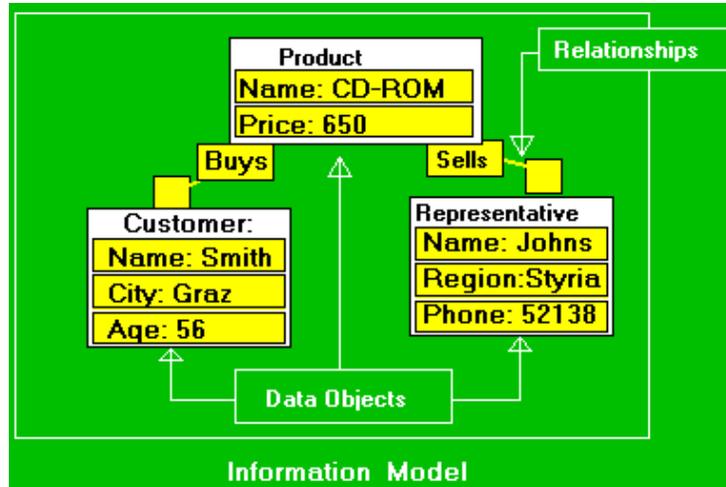
A record retains a value of the Data Base Key even if the record is modified until the record is finally deleted from the data base. In some ways, the data base key for the CODASYL record is like a social security number or a personal identification number.

### 1.4 Data Set

Normally, the information model consists of two main parts:

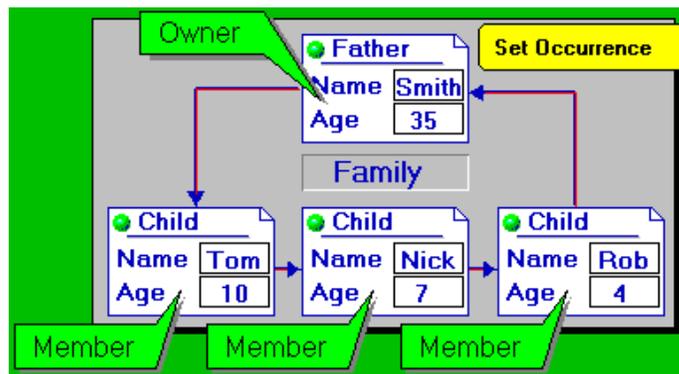
**Data Objects and Relationships.**

## Network (CODASYL) Data Model



In accordance with the Data Base Task Group (DBTG) proposals, each record directly corresponds to the concrete entity, but relationship between the records are implemented by means of a special logical construction. This logical construction is called a **Data Set** (or simply **Set**).

In the simplest case, each set (or more precisely, set occurrence) consists of records of two different types (e.g. Father and Child).



The data set has the following properties:

- 1. Each set includes exactly one record of the first type. This record is called an **Owner** of the set.
- 2. Each set may include 0 (i.e. an Empty set occurrence), 1 or N records of the same type. These records are called **members** of the data set.
- 3. All members within one set occurrence have a fixed order (are sorted).

There may be two or more sets consisting of records of the same types and describing the same relationship between records.

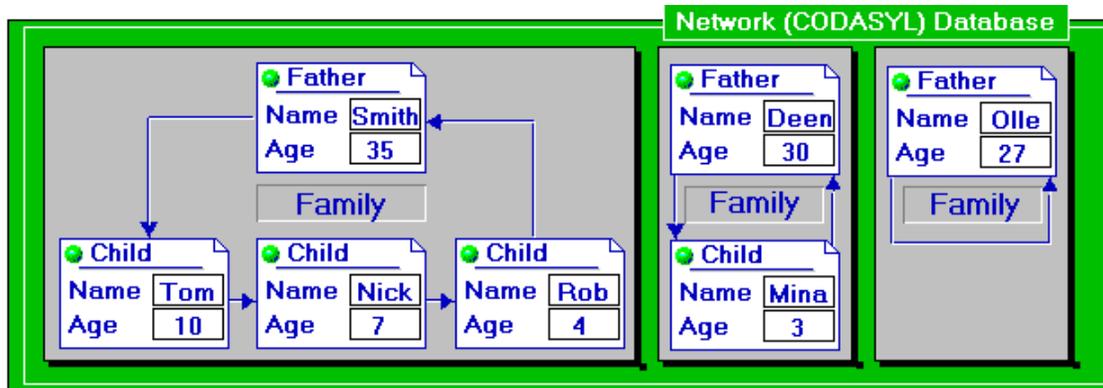
A collection of such set occurrences is called a **set type**. Each set type has a unique name.

## Network (CODASYL) Data Model

We can thus say that different set occurrences of the same set type describe different instances of a certain relationship between entities of the area being modelled. Thus, the main data structure types used in the network data model are:

- record type,
- set type.

In other words, according to the network data model the information within a database is arranged as a collection of record occurrences and a collection of set occurrences.



All record types and all set types must be described in the data base schema. The data base consists of record occurrences and of set occurrences of such types as were previously defined in the data base schema.

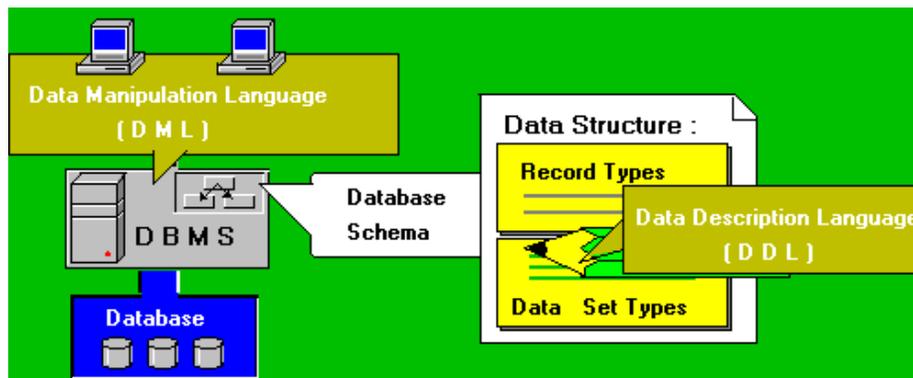
## 2 Bachmann Diagrams & Data Manipulation

### 2.1 Introduction

To create a data base, the Data Base Administrator (DBA) has to describe a data base structure (all record types and data set types) using the so-called **Data Description Language (DDL)**.

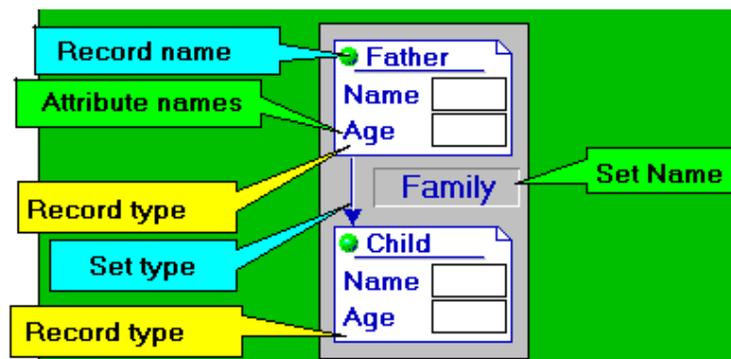
In addition to the definition of a data base structure (schema) the users have actually to create and maintain a data base using a **Data Manipulation Language (DML)**. In other words, the users must put new data into a data base and alter existing data in a data base. These facilities of a database management system are known as updating facilities or data update functions.

The two languages (**DDL** and **DML**) are very closely connected, i.e. in order to understand concepts of the DDL properly, we should know the main data manipulation functions.



### 2.2 Bachmann Diagram

There exists a useful and well-known graphic notation for the network data base schema. This notation is known as a **Bachman Diagram** ( or sometimes as a **Data Structure Diagram**).



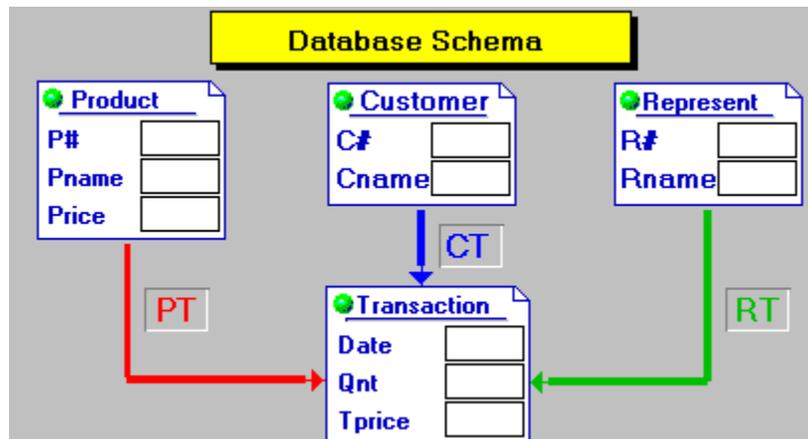
- 1. Each record type is depicted as a rectangle.
- 2. The rectangle contains a record type name and attribute names.
- 3. Each set type is depicted as an arrow.

## Network (CODASYL) Data Model

- 4. The arrow is directed to the member of the corresponding set type.

Consider a more complicated example. Suppose a company produces and sells computers. In this case all record types are evident:

- 1. Computer products that the company sells (**PRODUCT**);
- 2. Customers who buy the products (**CUSTOMER**);
- 3. Representatives who sell the products (**REPRESENT**);
- 4. Sales transactions (**TRANSACTION**);



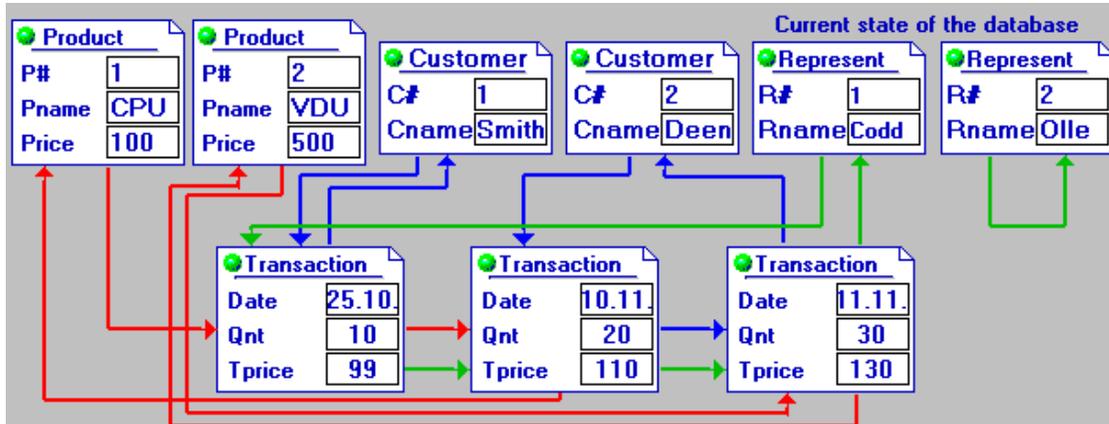
All set types are also evident:

- 1. Product and all transactions which include this company's product (**PT**);
- 2. Customer and all transactions which were done by this customer (**CT**);
- 3. Representative and all transactions which were done by this representative (**RT**);

A current state of the database might look as follows: The **Records**, for example, might be:

- 1. Computer products that the company sells (**PRODUCT**);
- 2. Customers who buy the products (**CUSTOMER**);
- 3. Representatives who sell the products (**REPRESENT**);
- 4. Sales transactions (**TRANSACTION**);

## Network (CODASYL) Data Model



The **Data Sets** might look as follows:

- 1. Products and all transactions which include these company's products (**PT**);
- 2. Customer and all transactions which were done by these customers(**CT**);
- 3. Representatives and all transactions which were done by these representatives(**RT**).

### 2.3 Data Updating Facilities

We have discussed only the first part of Network Data Model - the data description facilities. Each data model also includes particular data manipulation facilities - **Data Manipulation Language (DML)**. The data manipulation facilities of a concrete DML can be also divided into two parts:

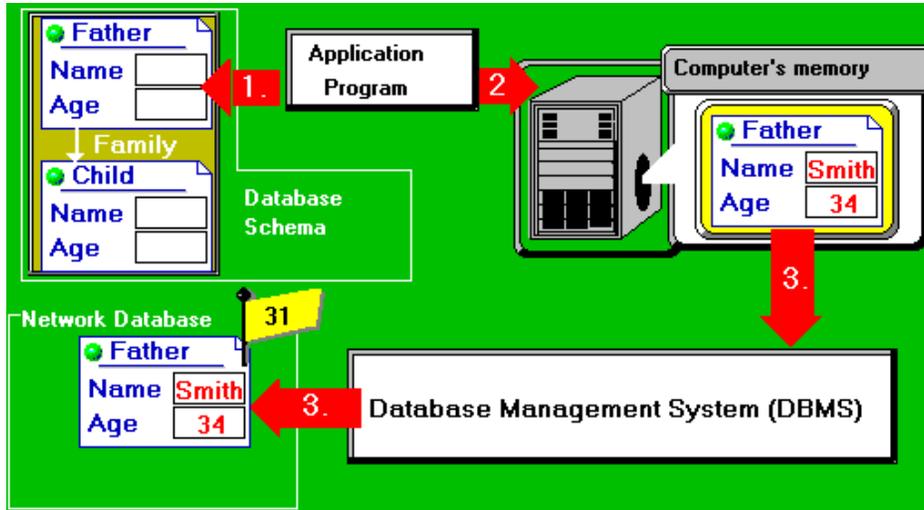
- (i) data update functions;
- (ii) data retrieve functions.

The main update functions of a Network data manipulation language are:

- 1. To store new occurrences of the record type declared in the current data base schema.
- 2. To modify existing occurrences of the record type declared in the current data base schema.
- 3. To delete existing occurrences of the record type declared in the current data base schema.
- 4. To insert existing occurrences of the record type declared in the current data base schema as a member of a certain data set into the exactly one occurrence of this data set.
- 5. To remove existing occurrence of the member of the data set from the occurrence of this data set.

**Putting a new record occurrence into a database:**

## Network (CODASYL) Data Model

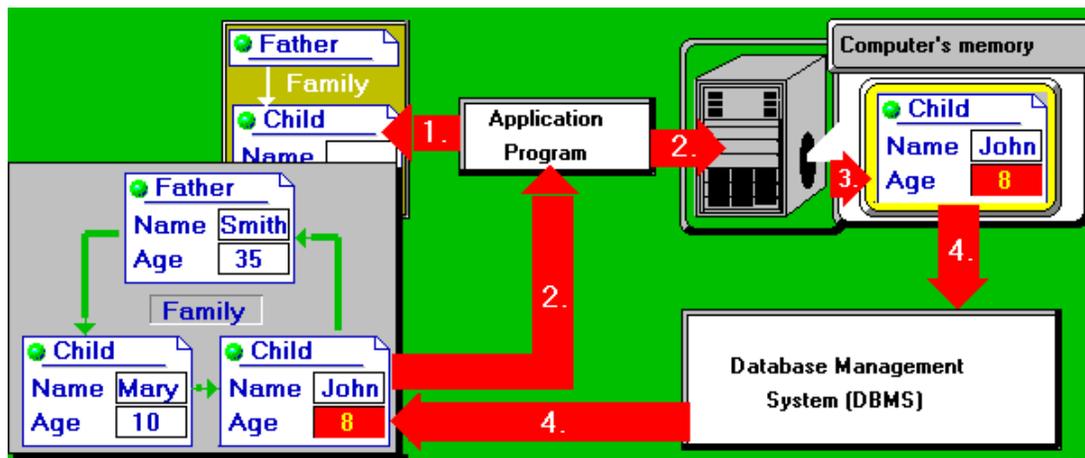


This process may be described as follows:

- Step 1: The application program chooses the record type.
- Step 2: The program prepares a new occurrence of the record type in the computer's memory.
- Step 3: The DBMS puts a new occurrence into the data base.

Note, when the owner of a certain data set is stored into the data base, the empty occurrence of this data set is constructed automatically in the data base.

### Modifying a record occurrence:



This process may be described as follows:

- Step 1: The application program chooses the record type.
- Step 2: The program retrieves an occurrence of the record type (data retrieving facility).
- Step 3: The program modifies one or more data items in the computer's memory.
- Step 4: The DBMS puts the record occurrence back into the data base.

### **Deleting a record occurrence:**

This process may be described as follows:

- Step 1: The application program chooses the record type.
- Step 2: The program retrieves or points out an occurrence of the record type (data retrieving facility).
- Step 3: The DBMS deletes the record which has been pointed out.

Note, if a member of a certain data set is deleted, it is also removed from this data set occurrence. An owner of a data set may be deleted if the corresponding occurrence is an empty data set.

### **Inserting a record occurrence into a set occurrence:**

This process may be described as follows:

- Step 1: The application program chooses the record type.
- Step 2: The program retrieves or points out an occurrence of the record type (data retrieving facility).
- Step 3: The program points out the occurrence of the data set (data retrieving facility).
- Step 4: The DBMS inserts the record occurrence which has been pointed out into the occurrence of the data set.

### **Removing a record occurrence from a data set:**

This process may be described as follows:

- Step 1: The application program chooses the record type.
- Step 2: The program retrieves or points out an occurrence of the record type (data retrieving facility).
- Step 3: The DBMS removes the record occurrence which has been pointed out from the occurrence of the data set.

## **2.4 Network Subschema**

The database schema defines the entire database which is stored and available to all users. An application program may need to view only some parts of the database, as well as to make some simple changes. A part of the database schema that is used by one or more application programs is called a **Database Subschema**.

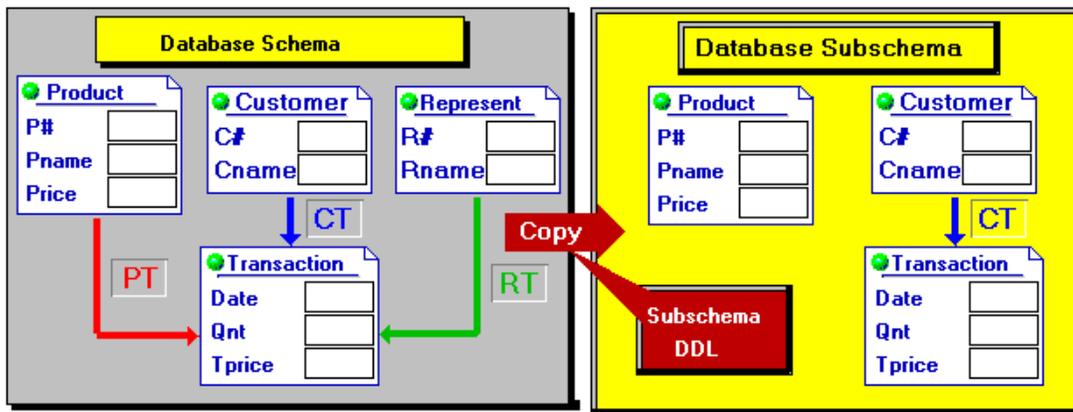
A database subschema is defined by a so-called **Subschema Data Description Language (Subschema DDL)**. In other words, subschema DDL allows a data administrator to determine which portions of a database (as declared in the database schema) are to be made available to the application program or programs.

In the simplest case, we can select a certain part of the database schema and consider it as a database subschema. Thus, the subschema DDL can be regarded as the only COPY

## Network (CODASYL) Data Model

statement which allows a database administrator to select a certain data structure type (for instance, set type, record type).

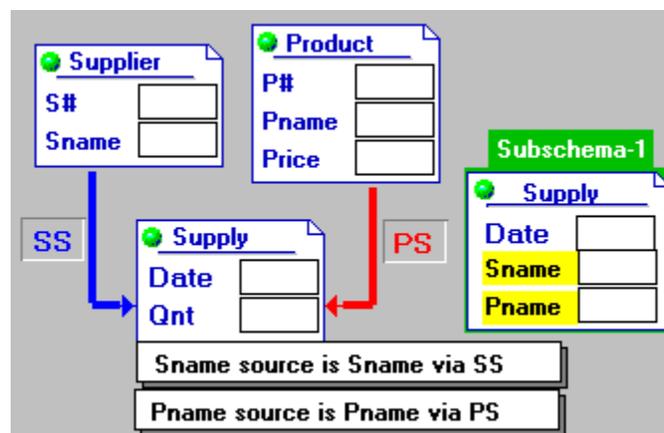
In this way a database administrator can eliminate unnecessary record types and set types from the network subschema.



Analogously, the database administrator can eliminate unnecessary data items.

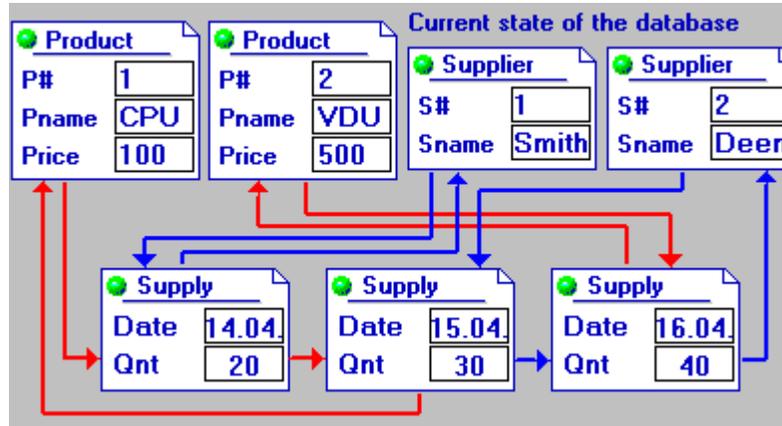
The **COPY <item name> ITEM** statement allows to select a certain field, and it can be regarded as elimination of unnecessary data items. The so-called virtual fields may also be declared as a part of the network subschema. Virtual fields are defined to be logically a part of a record, but not physically present in the record.

When we declare the virtual field, we define a source of this field, which is a field of the corresponding owner record. When we refer to a virtual field, DBMS obtains its value by following a link to the proper owner record and obtaining the source field from this record. Consider another subschema of the same database schema.



Suppose, current state of the database looks as follows:

# Network (CODASYL) Data Model



In this particular case, user view defined by the subschema, can be seen as the following database:

Supply	Date	14.04
	Pname	CPU
	Sname	Smith
Supply	Date	15.04
	Pname	CPU
	Sname	Deen
Supply	Date	16.04
	Pname	VDU
	Sname	Deen

### 3 CODASYL D D L

#### 3.1 Introduction

Recollect that Data Description Language (DDL) is a collection of statements for the description of data structure types.

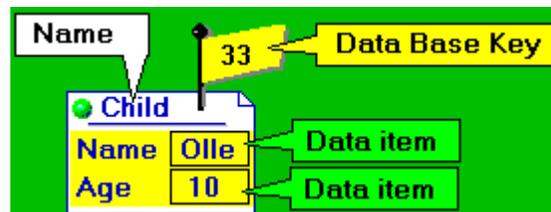
For the network data model, the main data structure types are:

- Record type;
- Set type.

Hence, the CODASYL DDL has to include statements for the description of the record types and the set types. The statements of the CODASYL DDL are called **clauses**.

#### 3.2 "Record" Clause

A CODASYL record includes a collection of attribute values (data items). In addition, each CODASYL record has a value of the Data Base Key. Note, that the CODASYL record type has a unique name as well.



Thus the definition of a record type must include:

1. Definition of a unique record name ("**Record Name**" sub-clause).
2. Definition of all attributes ("**Data Item**" sub-clause).
3. Definition of the so-called "**Location Mode**" ( how is a concrete value of DBK assigned to an occurrence of this record type).

Thus each CODASYL record type must be described in the following form:

- "Record Name" Sub-Clause
- "Location Mode" Sub-Clause
- "Data Item 1" Sub-Clause
- ...
- "Data Item n" Sub-Clause

The form of the "record name" sub-clause is:

**RECORD NAME IS <record name>**

- We are using the syntax expressions where:

## Network (CODASYL) Data Model

- Parts of the language are written in capital letters;
- Names to be provided by the user are written in lower case.
- Required words are underlined>. Words not underlined are so-called noise words which may be included to enhance the readability of the schema declaration. They may be omitted without loss of meaning.

All the following clauses, for example, are equivalent:

**RECORD NAME IS CUSTOMER**

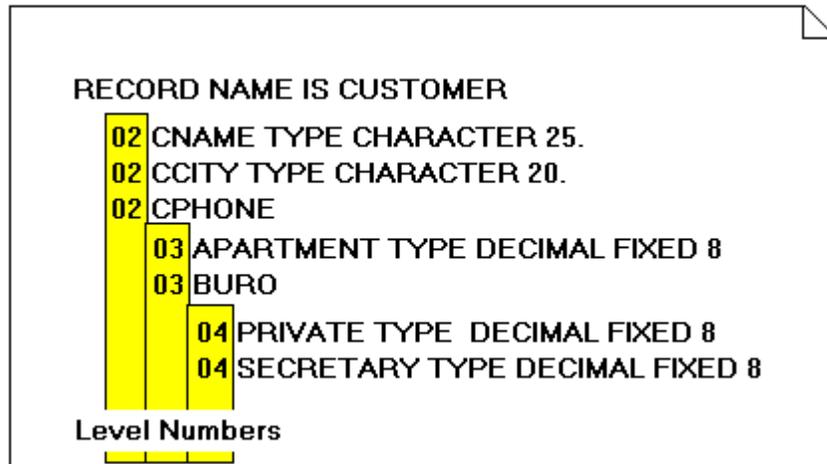
**RECORD IS CUSTOMER**

**RECORD CUSTOMER**

**RECORD NAME CUSTOMER**

The "data item" sub-clause defines an **elementary item**, a **group item** or a **table** similar to the DECLARE statements of the PL1, COBOL or other programming languages.

If some record type includes group items, a so-called **Level Number** should be put before each attribute name.



The OCCURS expression is used to define a table or a repeating group.

The form of the Occurs expression is:

**OCCURS{[integer],[data item]}TIMES**

In this syntax expression: The curly brackets (braces) imply that a choice of one of the options has to be made from the two or more what listed columnwise within brackets.

If the option [data item] is used, then it must be an elementary item in the record being defined, and it must also be of **TYPE DECIMAL FIXED** which implies that it takes only integer values.

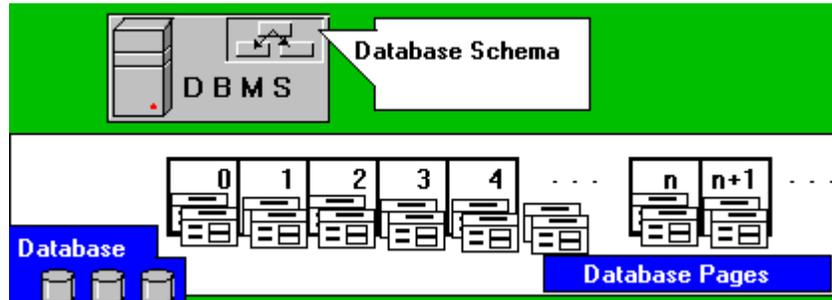
### 3.3 "Location Mode" Sub-clause

The "location mode" sub-clause defines the rules of assigning a data base key value to each record occurrence.

## Network (CODASYL) Data Model

Note that each record occurrence is assigned a data base key value at the time it is stored in the data base for the first time. In other words, the DBMS assigns a data base key value to record occurrences in accordance with the "location mode" sub-clause.

To understand the way the "location mode" sub-clause works in practice, recollect that record occurrences are resides within a particular database. The database is usually divided into a number of equal sized pages.



A typical "location mode" sub-clause defines a way to locate a page within the database. The record is then stored somewhere in that or a neighbor page where there is a space for it.

A database administrator (DBA) has a number of "location mode" options available. The DBA must choose exactly one for each record type.

The DBTG identified the following four options:

**SYSTEM**  
**CALC ...**  
**VIA ... SET**  
**DIRECT**

If the DBA chooses the option **SYSTEM**, then the DBMS "may use" any appropriate rule to store the record occurrences as quickly as possible (say, it can use a latest accessed database page residing in a system buffer).

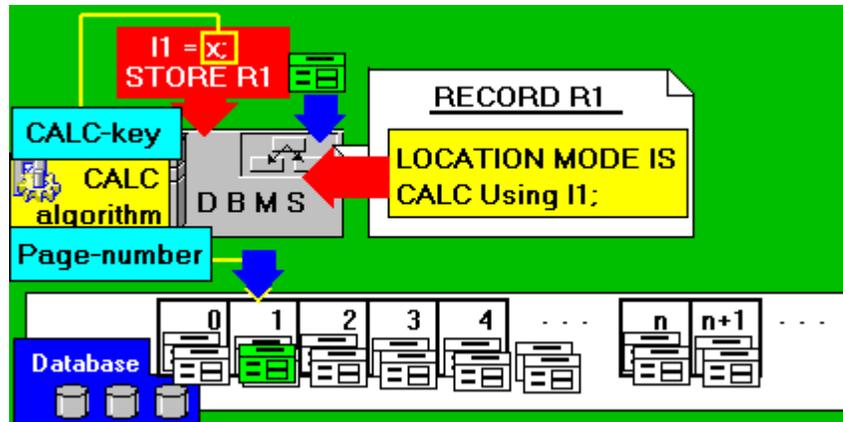
If the DBA chooses the option **CALC**, the form of the "location mode" sub-clause must be:

**CALC USING [data item, ...]**

where data item is an elementary item in the record being defined.

The word **CALC** is an abbreviation for calculation and the implication here is that, when a record is stored in the data base, its DBK value (a page where it is stored) is calculated from the concatenated value of one or more items named after **USING** in the "location mode" sub-clause.

These data items are called a **CALC-key**, and the algorithm of calculation is called **CALC-algorithm**.



Thus, the new records are stored on the page calculated by means of the CALC-algorithm if there is enough space for it. Otherwise a pointer to some other overflow page is stored and the record is stored in the overflow page pointer.

The page number may be generated for instance, as follows:

- 1. The value of the CALC-key, whether it is numeric or alphanumeric, is treated as one long bit string, effectively as an integer.
- 2. It is then divided by the number of pages in the database.
- 3. The remainder after the division is the page number.

When the DBA decides to select a location mode of CALC for a given record type, then he must also make a decision with respect to duplicate values of the CALC-key. More specifically, he must decide whether to allow or prohibit duplicate values of the CALC-key.

The form of the DUPLICATES expression is:

**DUPLICATES ARE {NOT} ALLOWED**

For example, if the CALC-key is the last name of a customer (CNAME), then normally the decision would be to allow duplicate values of the CALC-key. If the CALC-key is chosen as a unique product name, then the DBA must prevent the database from storing record occurrences with duplicate values of this data item.

The record with the CALC location mode may be easy to find in the database. To find this record, we need to know the value of the CALC-key.

In this case, we may repeat the calculation of the Data Base Key (page number), and retrieve record occurrence using the calculated value. The possibility of retrieving record occurrences by means of a value of the CALC-key is the main advantage of this "location mode" option.

The DBA should select a location mode of CALC for a certain record type, if the users need to retrieve occurrences of this record type using known values of one or more data items.

## Network (CODASYL) Data Model

Giving the record type a location mode of DIRECT means that the programmer has to generate data base key values manually (with an application program) and he can, therefore, control the position of a record occurrence in the data base.

If the DBA chooses the option DIRECT, the form of the "location mode" sub-clause must be

### **DIRECT USING < DBK-name >**

where <DBK-name > is a unique name of a data item.

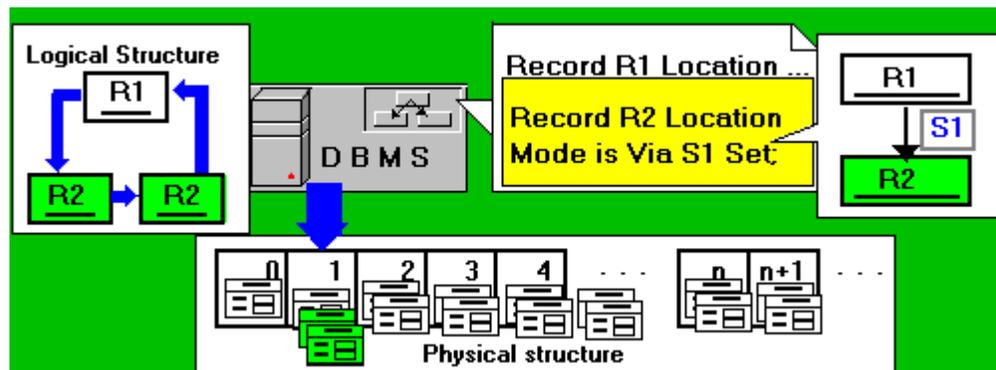
This item is the one which the programmer will use when he has somehow generated a data base key value which is not already used in the data base, and which should be assigned to a record he is storing therein.

If DBA chooses the option VIA, the form of the "location mode" sub-clause must be:

### **VIA <set name > SET**

In order to be able to use this location mode at all, the record type to which the location mode is being assigned must be a member in the <set name > set type.

The record type may be a member in other set types, but it is the membership in the set type named which dictates the meaning of the location. The essence of choosing a location mode of VIA is to cause a record to be stored physically close to the owner of the set occurrence (in the same page if there is space for it ).



### 3.4 Example

Consider the following Example of a Network Database: .



**Record Name is Customer  
Location Mode is Calc Using Cname  
Duplicates are not Allowed**

Network (CODASYL) Data Model

**Cname Type character 25.**  
**City Type character 20.**

**Record Name is Product**  
**Location Mode is Calc Using Pname**  
**Duplicates are Allowed**  
**Pname Type character 25.**  
**Price Type Decimal Fixed 4.**

**Record Name is Transaction**  
**Location Mode is System**  
**Date Type Decimal Fixed 6.**  
**Qnt Type Decimal Fixed 4.**

## 4 CODASYL D D L (Part 2)

### 4.1 Introduction

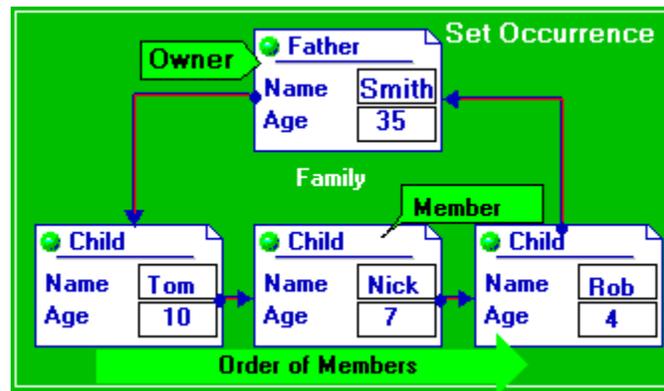
Recollect that the main data structure types being used in the network data model are: **record type** and **set type**. All record types and all set types must be described in the data base schema.

The data base consists of **record occurrences** and of **set occurrences** of such types as are previously defined in the data base schema.

The network DDL includes:

- - the record clause to define record types;
- - the set clause to define set types;

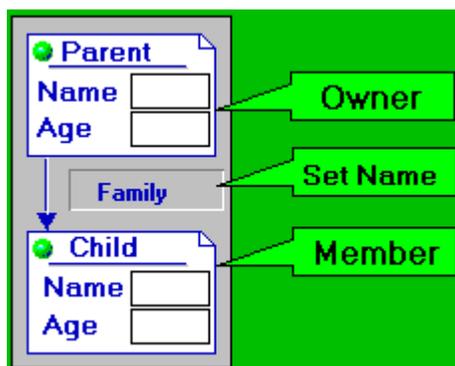
A **data set** is a special data structure which includes record occurrences of two different types.



The data set has certain properties:

1. Each set includes exactly one record of the first type. This record is called an **owner** of the data set.
2. Each set may include 0 (i.e. an Empty set occurrence), 1 or N records of the same type. These records are called **members** of the data set.
3. All members within one set occurrence have a fixed order (are sorted).

Thus, in order to define a set type we have to define:



1. an unique **name** of the set type;
2. a name of the **owner** of this set type;
3. a name of the **member** of this set type;
4. an **order** of members within a set occurrence;
5. some **additional properties** of the set type.

#### 4.2 Set clause

The **SET NAME** sub-clause defines the unique name of the set type.

The form of the sub-clause is:

**SET NAME IS [set name]**

For instance,

**Set Name is Family . . .**

The **OWNER** sub-clause names the owner record type.

The form of the sub-clause is:

**OWNER IS [record name]**

Note that the [record name] must be defined by the RECORD clause before-hands.

For instance,

**Record Name is Parent ...**

**Set Name is Family**

**Owner is Parent**

The **MEMBER** sub-clause names the member record type.

The form of the sub-clause is:

**OWNER IS [record name]**

Note that the [record name] must be defined by the RECORD clause before-hands.

For instance,

**Record Name is Parent ...**

**Record Name is Child ...**

**Set Name is Family**

**Owner is Parent**

**Member is Child**

A set occurrence can be viewed as a ring consisting of the owner and each of the members. Thus, each member of a set occurrence contains the reference (pointer) to the "next" member or owner.

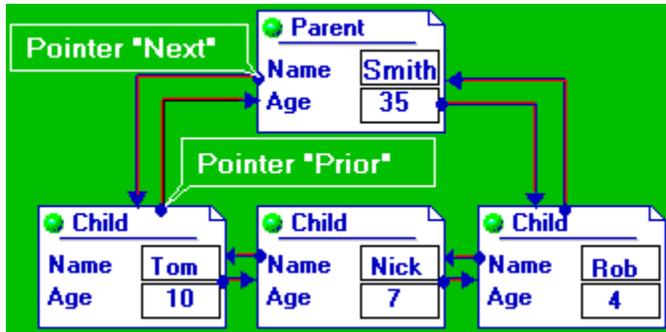
The set type may be defined with the option:

**SET IS PRIOR PROCESSABLE.**

In this case, each member and owner of the set occurrence contains the additional reference (pointer) to the "prior" member or owner.

For instance,

Network (CODASYL) Data Model



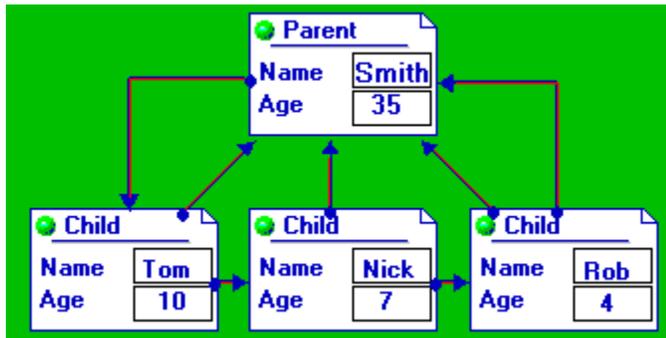
Record Name is Parent ...  
 Record Name is Child ...  
 Set Name is Family  
 Set is Prior Processable  
 Owner is Parent  
 Member is Child

A set type may be defined with the option:

**SET IS LINKED TO OWNER.**

In this case, each member of the set occurrence contains an additional reference (pointer) to the owner.

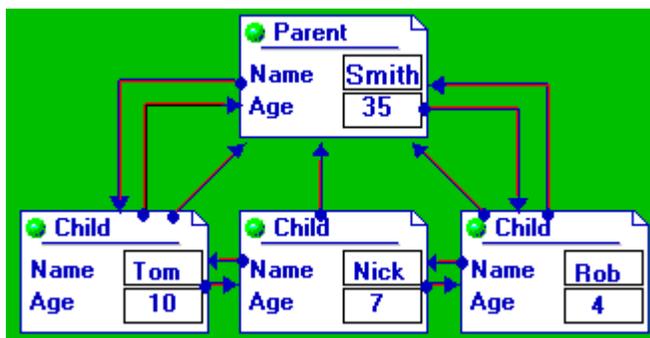
For instance,



Record Name is Parent ...  
 Record Name is Child ...  
 Set Name is Family  
 Owner is Parent  
 Member is Child Linked to Owner

Note that the option **PRIOR PROCESSABLE** and the option **LINKED TO OWNER** may be used in one set clause.

For instance,



Record Name is Parent ...  
 Record Name is Child ...  
 Set Name is Family  
 Set is Prior Processable  
 Owner is Parent  
 Member is Child Linked to Owner

Some DML commands can be executed more efficiently if the set type is defined with these options.

#### 4.3 Insertion sub-clause

The **INSERTION** sub-clause specifies how member record occurrences are initially placed in a set occurrence.

The form of the **INSERTION** sub-clause is:

**INSERTION IS {AUTOMATIC / MANUAL}**.

The **AUTOMATIC** specification indicates that each time a new occurrence of the member record is stored in the database, it is automatically inserted in this set.

For instance,

**Record Name is Parent ...**  
**Record Name is Child ...**  
**Set Name is Family**  
**Insertion is Automatic**  
**Owner is Parent**  
**Member is Child**

The **MANUAL** specification indicates that member records are inserted in the set occurrence manually, i.e., using an additional **INSERT** command.

For instance,

**Record Name is Parent ...**  
**Record Name is Child ...**  
**Set Name is Family**  
**Insertion is Manual**  
**Owner is Parent**  
**Member is Child**

#### 4.4 Retention sub-clause

The **RETENTION** sub-clause concerns the removal of member record occurrences from set occurrences.

The form of the **RETENTION** sub-clause is:

**RETENTION IS {MANDATORY / OPTIONAL}**.

The **OPTIONAL** specification allows removing of a member occurrence from the set occurrence.

For instance,

**Record Name is Parent ...**  
**Record Name is Child ...**  
**Set Name is Family**  
**Insertion is Automatic Retention is Optional**  
**Owner is Parent**  
**Member is Child**

The **MANDATORY** specification indicates that once a member occurrence is placed in the set occurrence it may not be **REMOVED** from the set occurrence without actually deleting the record occurrence.

For instance,

**Record Name is Parent ...**  
**Record Name is Child ...**

**Set Name is Family**  
**Insertion is Automatic Retention is Mandatory**  
**Owner is Parent**  
**Member is Child**

Note that the MANDATORY specification does not prevent an actual deleting of such member records from the database.

#### 4.5 Order sub-clause

The **ORDER** sub-clause specifies the order in which member record occurrences may be presented sequentially to an application program (i.e. is sorted within a set occurrence). Note that set occurrences are made as a result of sequence of the **STORE/INSERT** commands.

Thus, the **ORDER** sub-clause defines the exact place where a new member occurrence is to be inserted into the set occurrence. The form of the **INSERTION** sub-clause is:

**ORDER IS {FIRST / LAST / NEXT / PRIOR / SORTED BY ...}**.

If the **ORDER** sub-clause is defined as: **ORDER IS FIRST** then a new member occurrence is inserted as the "first" member of the set occurrence.

For instance,

**Record Name is Parent ...**  
**Record Name is Child ...**  
**Set Name is Family**  
**Owner is Parent**  
**Order is First**  
**Insertion is Automatic Retention is Optional**  
**Member is Child**

If the **ORDER** sub-clause is defined as: **ORDER IS LAST** then a new member occurrence is inserted as the "last" member of the set occurrence.

For instance,

**Record Name is Parent ...**  
**Record Name is Child ...**  
**Set Name is Family**  
**Owner is Parent**  
**Order is Last**  
**Insertion is Automatic Retention is Mandatory**  
**Member is Child**

If the **ORDER** sub-clause is defined as: **ORDER IS NEXT** then a new member occurrence is inserted as the "next" member after the most recently accessed (i.e. **current**) member or owner of this set occurrence.

For instance,

**Record Name is Parent ...**  
**Record Name is Child ...**  
**Set Name is Family**  
**Owner is Parent**  
**Order is Next**  
**Insertion is Manual Retention is Optional**  
**Member is Child**

## Network (CODASYL) Data Model

If the ORDER sub-clause is defined as: **ORDER IS PRIOR** then a new member occurrence is inserted as the "prior" member before the most recently accessed (i.e. **current**) member or owner of this set occurrence.

For instance,

**Record Name is Parent ...**  
**Record Name is Child ...**  
**Set Name is Family**  
**Owner is Parent**  
**Order is Prior**  
**Insertion is Automatic Retention is Optional**  
**Member is Child**

The option:

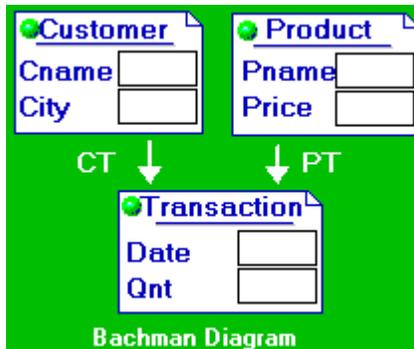
**ORDER IS SORTED BY [attribute] {ASCENDING/DESCENDING}**  
specifies the order of the set to be sorted based on keys stated as part of this option.

For instance,

**Record Name is Parent ...**  
**Record Name is Child ...**  
**Set Name is Family**  
**Owner is Parent**  
**Order is Sorted by Child.Age Descending**  
**Insertion is Automatic Retention is Optional**  
**Member is Child**

### 4.6 Illustrative Example

Consider the following Example of a Network Database:



/\* Record Clauses: \*/

**Record Name is Customer**  
**Location Mode is Calc Using Cname**  
**Duplicates are not Allowed**  
**Cname Type character 25.**  
**City Type character 20.**

**Record Name is Product**  
**Location Mode is Calc Using Pname**  
**Duplicates are Allowed**  
**Pname Type character 25.**  
**Price Type Decimal Fixed 4.**

**Record Name is Transaction**

Network (CODASYL) Data Model

**Location Mode is VIA CT**  
**Date Type Decimal Fixed 6.**  
**Qnt Type Decimal Fixed 4.**

*/\* Set Clauses: \*/*

**Set CT Owner is Customer;**  
**Order is Sorted by Key Date Descending**  
**Member is Transaction;**  
**Insertion is Automatic;**  
**Retention is Mandatory.**

**Set PT Owner is Product;**  
**Order is Last**  
**Member is Transaction;**  
**Insertion is Automatic;**  
**Retention is Mandatory.**

## 5 Data Manipulation Facilities

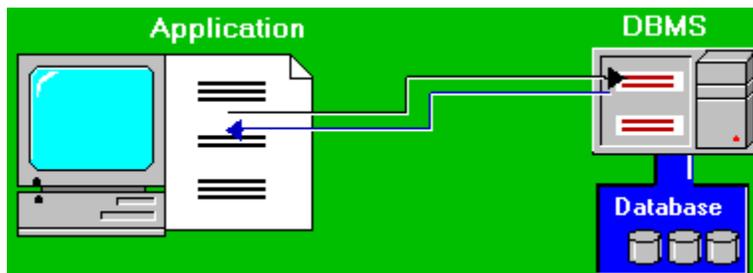
### 5.1 Introduction

In the CODASYL approach, all programs are written in a host language augmented by the commands of the Data Manipulation Language, such as

- - FIND (locate a described record occurrence),
- - GET (read a record occurrence from the database),
- - STORE (put a record occurrence into the database)

and so on.

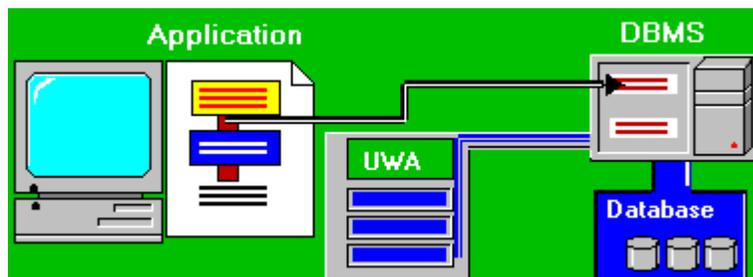
Thus, at least two different programs (i.e. an **application program** and **DBMS**) run simultaneously and have to exchange data.



Normally, an application program sends to the DBMS a request (DML operator) and additional parameters (data), and waits for a response. DBMS interprets the DML statements, retrieves (modify) data and sends results to the application program.

### 5.2 Application environment

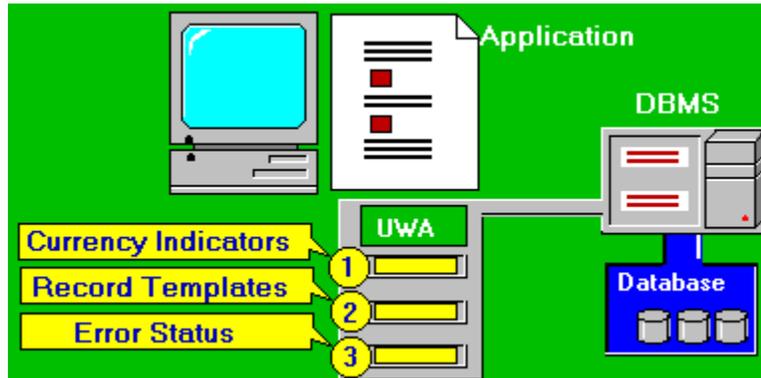
Actually, DBMS creates a special environment - a so-called **User Working Area (UWA)** for each of simultaneously running applications.



A DML command (request) is interpreted as follows:

- an application accesses UWA and set up some parameters by means of host language statement;
- an application program sends a request (DML operator) to the DBMS;
- the DBMS interprets the operator using the previously installed set of parameters and modifies data in UWA;

In the UWA it is found space for three kinds of data:



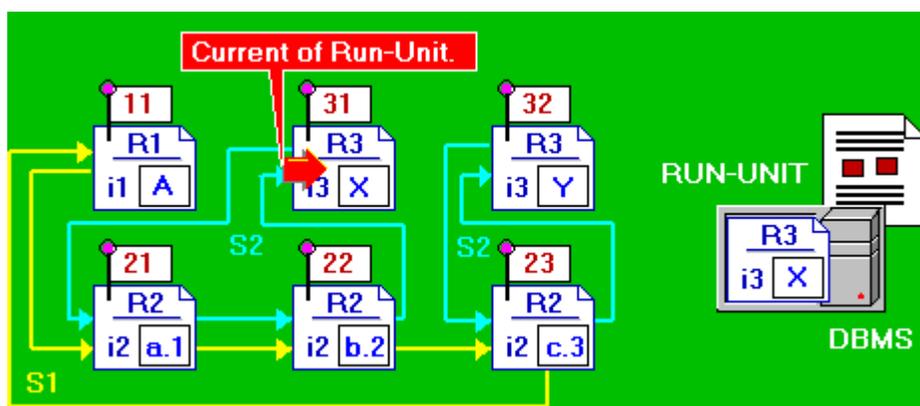
- 1. **Currency indicators**, which are pointers, or references, to certain record occurrences in the database;
- 2. **Templates** for various record types.
- 3. **Error status**, which is a special variable to inform an application program either the last DML operator was successfully executed, or not.

The data manipulation facilities of the Codasyl model are centered on a concept of what are called currency indicators and concept of navigation.

In a conventional file processing, a latest file record read by an application may be assumed to be the current record of that file as it is available to the program for processing.

### 5.3 Currency indicators

Analogously, the concept of the current record may be extended in the Codasyl model as follows:

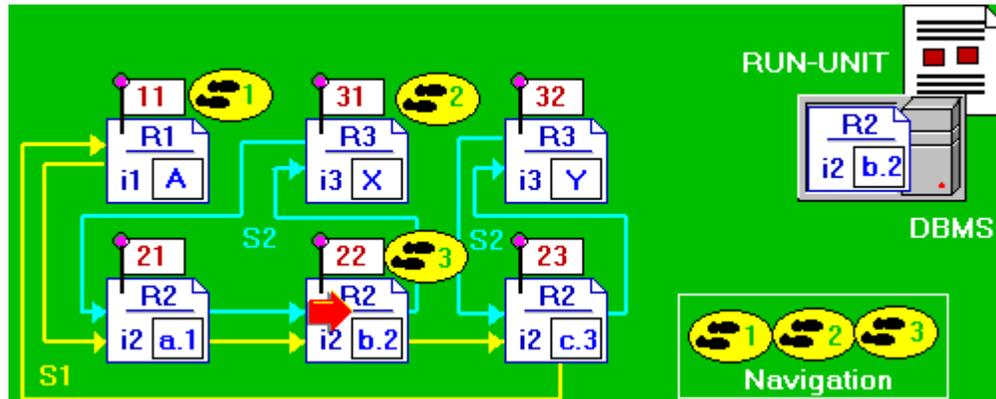


There exists only one record occurrence within the database currently available to the **RUN-UNIT** for processing. This record occurrence is called a **current of run-unit**.

Note that the term "**run-unit**" means "an application program" in the Codasyl proposals.

Basically, when a record occurrence is selected from the database by a DML operator FIND or stored in the database by a DML operator STORE, then it becomes the **current record of run-unit**.

The sequence of record occurrences which were current records of the run-unit during the execution of a certain application program is called a **navigation**.



The concept of the current record may be further extended:

- 1. The most recently accessed record occurrence, of any type whatsoever, is called the "**current of run-unit**".  
Note, there may exist the only current of run-unit.
- 2. For each record type (say, R1), the most recently accessed record occurrence of this type is referred to as the "**current of R1**".  
Note, there may exist two or more currents of different record types (i.e. one current record of each record type).

By analogy we can introduce a so-called "**current of set type**":

For each Codasyl set type (say S1), consisting of owner record type (R1) and member record type (R2), the most recently accessed record of type R1 or R2 is called the "**current of S1**".

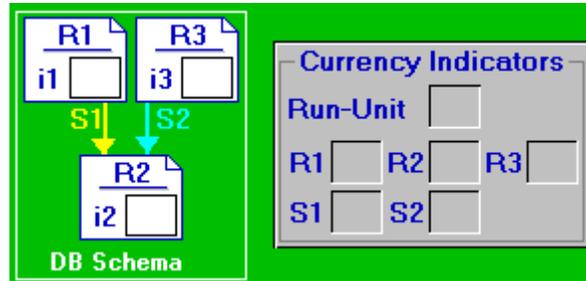
Note, there may exist two or more currents of different set types (i.e. one current record of each set type).

Note that sometimes the current of set will be an owner, and sometimes it will be a member.

In fact, currency indicators are special variables which contain Data Base Keys of different current record occurrences in the internal format.

Consider the following Database Schema:

## Network (CODASYL) Data Model



In this particular case, the UWA includes the following currency indicators:

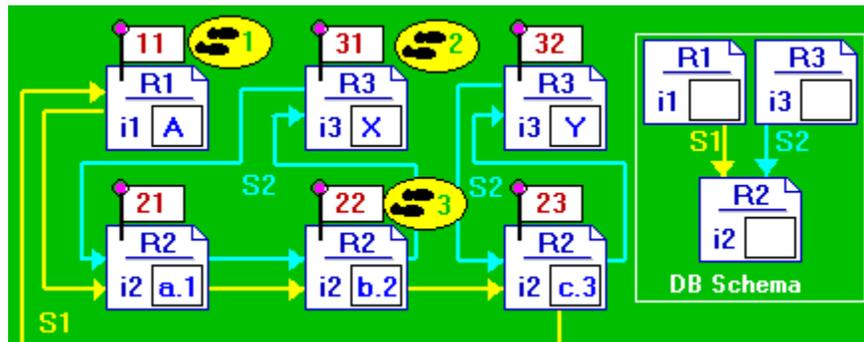
- 1. One currency indicator for the **run-unit**;
- 2. One currency indicator for each **record type**, indicating the current record of that type;
- 3. One currency indicator for each set type, indicating the current set occurrence of that type by "pointing" at the referenced record occurrence which is either the owner or a member of the given set.

Currency indicators always change on execution of some DML operator.

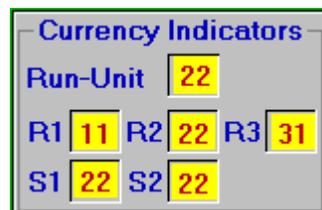
More precisely, when a new record occurrence of any type is found, it becomes:

- - the current record of **run-unit**;
- - the current record of its **record type**;
- - the current record of **all sets** in which it participates as either owner or member. Thus, the currency of many sets may be affected.

Let us consider the following navigation through the database.



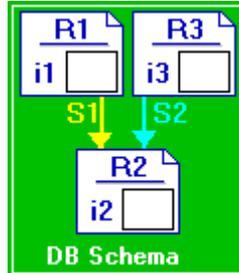
In this particular case, the navigation's effect on currency pointers looks like the following:



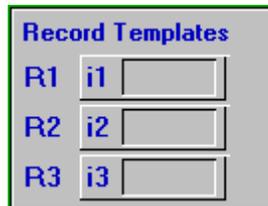
### 5.4 Record templates

The template for a record type consists of space for each field of this record type.

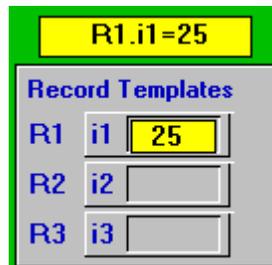
Consider the following Database Schema:



In this particular case, the UWA includes the following record templates:



That space is referred to as [record name]. [attribute name] (or just [attribute name] if the field name is unique) in application programs.



We shall use a "pidgin" version of the Pascal programming language. Note that operators of the host language are provided with comments. Hence, no knowledge of the Pascal is required.

For instance, the operator

**R1.i1=25**

changes the i1 field of the template

Similarly, the fields of record templates can be used as a source of data for further processing.

**R1.i1= R1.i1 + 25;**

**Print (R1.i1);**

A special GET command reads a record occurrence from database into an appropriate template. After that the contents of the template (data items) may be used in an application program.

Similarly, a record is stored into the database only after assembling the record occurrence in the template for its type, and the STORE operator simply copies the contents of the template into the database.

## 5.5 Error Status

The "last" part of the UWA is called an **ERROR STATUS**. In fact, error status is a special variable of type INTEGER.

This variable can be referenced in the application program by the name **Error\_Status**. The error status, upon return from a DML command, will contain a value of "0" (zero) if the command is successfully executed. A nonzero code represents an error condition, with the value for the code indicating the nature of the error.

There is generally a distinction between error situations that we expect to happen as a part of our normal processing, and error situations that are totally unexpected.

An example of an expected error situation occurs when we sequentially traverse through a set occurrence and reach the end of the set occurrence. Such an error situation usually means that we should proceed to another part of our program to continue processing.

An example of an unexpected error condition is an input/output error (for example, bad parity detected). Such an error situation usually means that DBMS should interrupt the application program execution.

The error status is usually used to notify an application program when an expected error situation occurs.

## 6 CODASYL DML (Part 1)

### 6.1 Introduction

In the network data model the process of data retrieving can be seen as step-by-step reading of record occurrences from a **database** to **UWA**.

Reading a record occurrence is a two stage process

- 1. a desired record occurrence is located using a sequence of FIND statements (i.e. it becomes the current of run-unit).  
Note, since **FIND** statements alter currency indicators and do not affect record templates, nothing has been copied into the record template at this point.
- 2. the desired record occurrence is copied into the record template using a special GET command.

The **GET** command always copies the current of run-unit into a template for whatever record type which is the current of run-unit.

The form of the GET command is:

**GET [record name]**

Thus, for debugging purposes, we can append the record type to the GET command. For example, the command **GET R2** will copy the current of run-unit into the R2 template, if the current of run-unit is a R2 record. Otherwise, the system will warn the user of an error when the GET R2 command is executed (i.e. the DBMS returns a nonzero value of the error status).

The **FIND** command in the Codasyl proposals is really a collection of different commands, distinguished by the keywords following FIND. These commands have the common purpose of locating a particular record occurrence by some designated strategy.

Note that the term "locate a record occurrence" in the Codasyl proposals is perceived as "make a record occurrence the current record of the run-unit and area in which it resides; make it also the current record of its record type and of all sets in which it participates as either owner or member".

The variety of FIND statements is extensive, and here we shall consider only the following useful subset of the possibilities:

- 1. Find a record given its database key, i.e., a pointer to the record occurrence.
- 2. Find a record given a value for its CALC-key.
- 3. Visit all members of a set occurrence in turn.
- 4. Scan a set occurrence for those member records having specified values in certain fields.
- 5. Find an owner of a given Codasyl set.

## 6.2 Finding a Record Directly

The first two kinds of FIND statements access record occurrences by a "key", either the database key or the CALC-key.

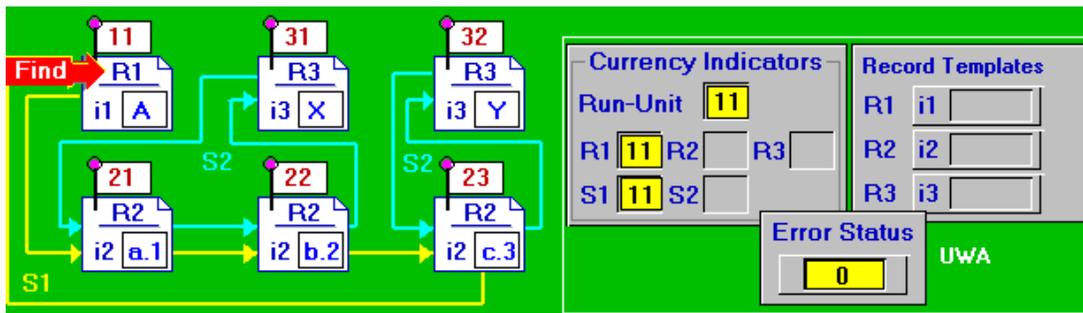
To access by the database key we write:

**Find [record name] Record by Database Key [variable]**

where the [variable] is a program's variable that has previously been given a database key as value.

For example,

```
X=11; Find R1 Record by Database Key X;
Get R1;
```

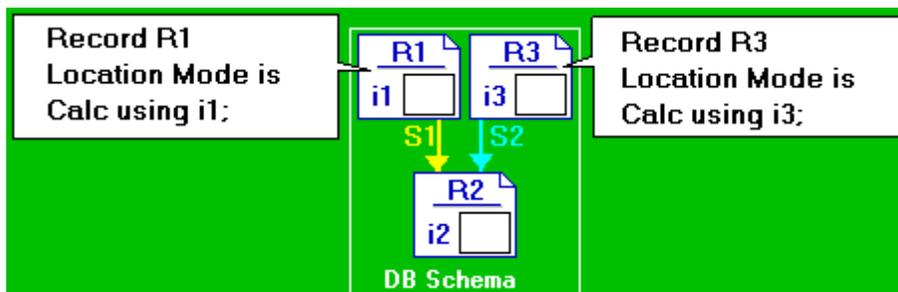


Obviously, we have to be careful with this command and check if the FIND operator was successfully executed. Perhaps, there is no such a record occurrence having that database key.

For example,

```
X=10; Find R2 Record by Database Key X;
If Error_Status=0 then Get R2;
Else Print "Not Found !";
```

In order to find a record directly using known values of its fields, such fields should be defined as so-called CALC - keys in the database schema.



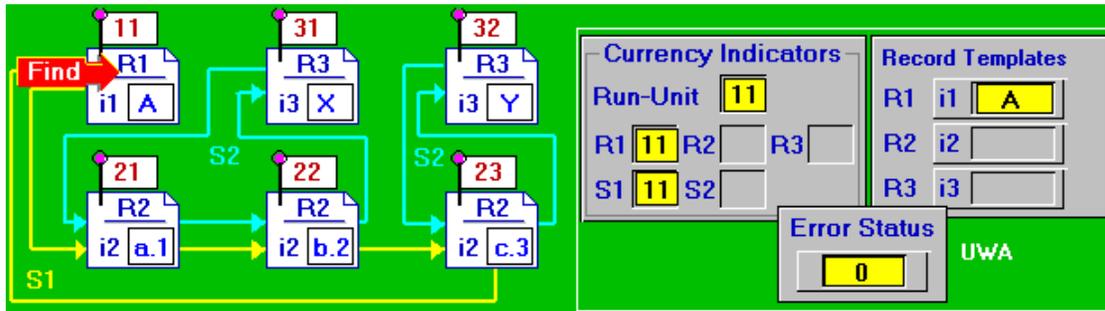
To find a record given values for its CALC-key fields, we "pass" these values in the corresponding fields of the template.

Then we issue the command:

**FIND [record name] RECORD BY CALC-KEY.**

For example,

R1.i1="A";  
Find R1 Record by Calc-Key



Note that there can be more than one record occurrence with the same value of the fields in the CALC-key, if the record type was defined with the option **DUPLICATES ARE ALLOWED**.

To find all the record occurrences of a given type (say, R1) with a given value for the CALC-key we can find the first such record occurrence as in the previous case.

R1.i1="A"; Find R1 Record by Calc-Key;

And then we can find additional record occurrences with the same CALC-key by executing, in a loop,

**FIND DUPLICATE [record type] RECORD BY CALC-KEY.**

For example,

R1.i1="A"; Find R1 Record by Calc-Key;  
Find Duplicate R1 Record by Calc-Key;

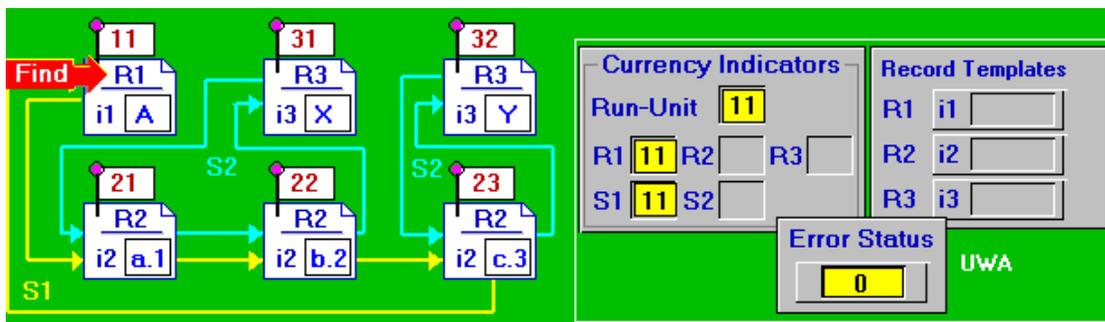
When performing any sort of scan, we must be prepared to find no record occurrences matching the specification:

For example,

R1.i1="A"; Find R1 Record by Calc-Key;  
Loop: Find Duplicate R1 Record by Calc-Key;  
If Error\_Status=0 then begin;  
... Go to Loop; end;

### 6.3 Scanning a Set Occurrence

To begin, suppose we have a current set occurrence for some Codasyl set (say, S1). Recall that the set occurrence can be viewed as a ring consisting of the owner (DBK=11) and each of the members (DBK=21, DBK=22 and DBK=23).



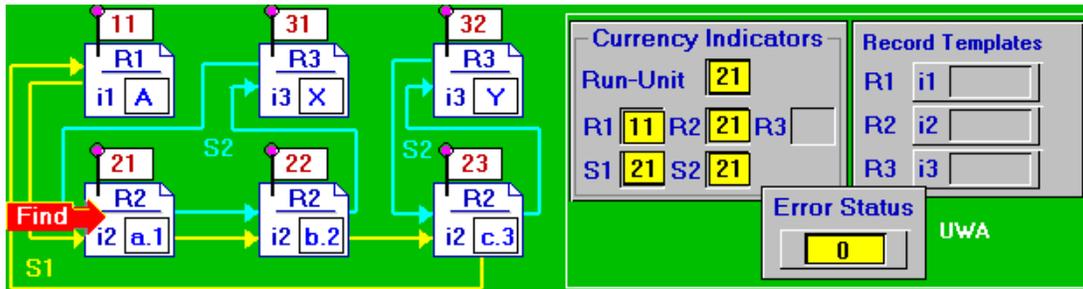
The statement:

**Find Next [record name] Record In [set name] Set**

goes one position around the ring from the current of [set name].

For example,

**Find Next R2 Record In S1 Set;**



The FIND NEXT command can be repeated as many times as we like, taking us around the ring for the set occurrence.

For example,

**Loop: Find Next R2 Record In S1 Set;  
If Error\_Status=0 then goto Loop;**

The operator FIND NEXT returns a non-zero value to the error status if the next record is not of the [record type], so we fail when we try to get back to the owner.

We may also issue the command:

**Find First [record name] Record In [set name] Set**

to get the first member record of the current [set name] set.

For example,

**Find First R2 Record In S1 Set;** The operator **FIND FIRST** returns a non-zero value to the error status if there are no members of the current set occurrence. Otherwise, we can continue around the ring with a loop containing a FIND NEXT command, as above.

For example,

**Find First R2 Record In S1 Set;  
Find Next R2 Record In S1 Set;**

We can also scan a set occurrence in the "reverse" direction (i.e. backwards from the last member up to the first member).

The statement

**Find Prior [record name] Record In [set name] Set**

goes one position back from the current of [set name].

For example,

**Find Prior R2 Record In S1 Set;**

The statement **FIND PRIOR** returns a nonzero value to the error status if the prior member is not of the [record type], i.e., when we get back to the owner.

The statement

**Find Last [record name] Record In [set name] Set**

goes to the last member record of the current of [set name].

For example,

**Find Last R2 Record In S1 Set;**

The statement **FIND LAST** returns a nonzero value to the error status if there are no members of this set occurrence.

Note that the statements **FIND LAST** and **FIND PRIOR** can be efficiently used if the set type is defined with the option

**SET IS PRIOR PROCESSABLE**

The next type of **FIND** statement also scans the members of a set occurrence, but it allows us to look at records only with specified values in certain fields (so-called, **search keys**).

The values for these fields (search keys) are stored in the template for the member record type before using the statement **FIND**.

For example,

**R2.i2="a.1";**

To find the first member record having the desired values we write:

**Find [record type] Record In [set name] Set Using [field list]**

For example,

**R2.i2="a.1";**

**Find R2 Record In S1 Set Using i2;**

To find subsequent records in the same set occurrence with the same values of selected fields we can say:

**Find Duplicate [record type] Record In [set name] Set Using [field list]**

For example,

**R2.i2="a.1";**

**Find R2 Record In S1 Set Using i2;**

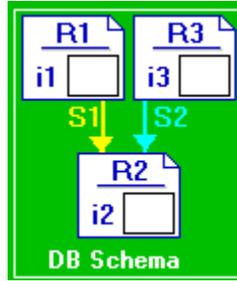
**Find Duplicate R2 Record In S1 Set Using i2;**

This operator **FIND** returns nonzero value to the error status if the current of set is the last member of the current set occurrence having the specified values.

#### **6.4 Finding an Owner**

Normally, the statement **FIND OWNER** is used if a record occurrence can be accessed through two or more different set occurrences.

## Network (CODASYL) Data Model



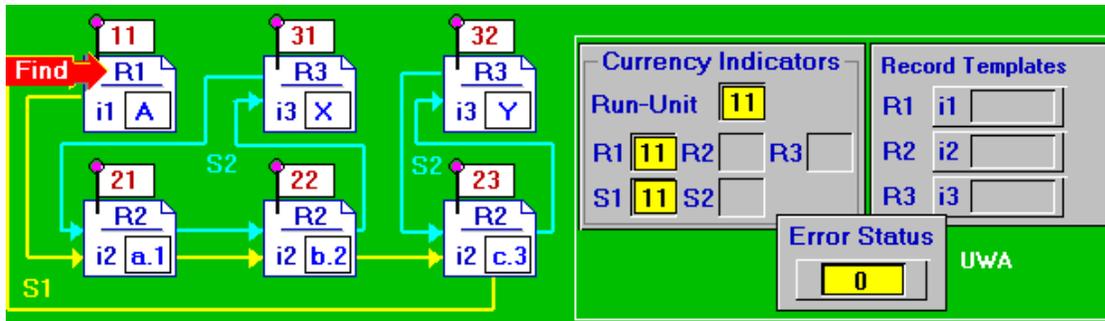
Note that the record type **R2** is declared as a member of two set types (**S1** and **S2**).

The statement:

**Find Owner Of [set name] Set;**

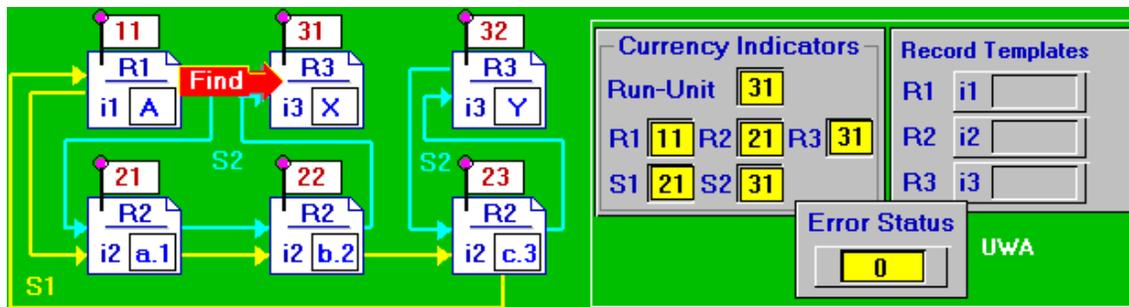
locates an owner of the current set.

For example,



**Find Next R2 Member of S1 Set;**

**Find Owner of S2 Set;**



Note, the operator **FIND OWNER** can be efficiently used if the set type is defined with the option:

**LINKED TO OWNER**

## 7 CODASYL DML (Part 2)

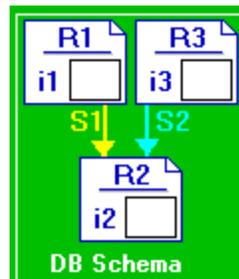
### 7.1 Introduction

In addition to the FIND and GET operations, used for database navigation and record retrieval, the Codasyl proposals include commands which perform update operations:

- STORE a new record occurrence;
- INSERT a member into a set occurrence;
- REMOVE a member from a set occurrence;
- DELETE a current record;
- MODIFY a current record.

Because of the many different options that can be coded by Codasyl DDL, database updates are very complex operations. Their semantics strongly depends on the database structure description (database schema).

Consider the following database and the following database schema.



In the following examples we assume that the set **S1** is defined with the options:

**INSERTION IS AUTOMATIC,  
RETENTION IS MANDATORY and  
ORDER IS LAST.**

We assume also that the set **S2** is defined with the options:

**INSERTION IS MANUAL,  
RETENTION IS OPTIONAL and  
ORDER IS FIRST.**

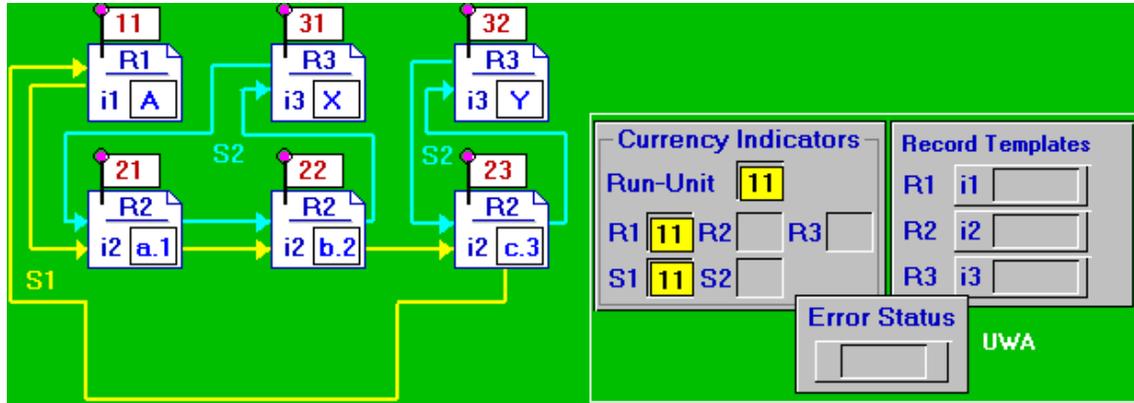
### 7.2 STORE Operator

To store a new record occurrence of a certain type (say, R2) into a database we have to:

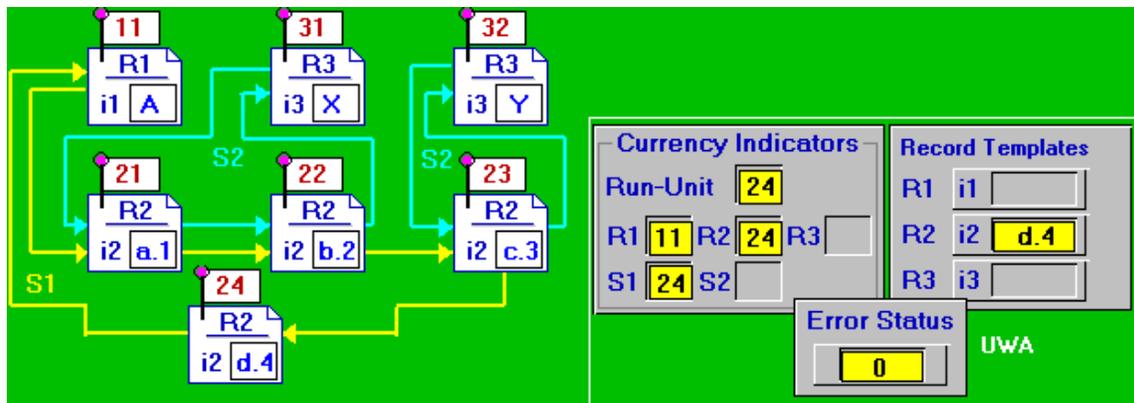
1. create a record occurrence in UWA;
- 2. issue the command: **STORE [record\_type] RECORD;**

For example,

## Network (CODASYL) Data Model



R2.i2="d.4";  
Store R2;



The STORE operator:

- 1. adds a new record occurrence to the collection of record occurrences of [record\_type];
- 2. assigns a unique database key (DBK) to the new occurrence;  
Note, the database key of the new record occurrence (and, hence, its position within the database) depends on the **LOCATION MODE** sub\_clause of the record type description.
- 3. automatically inserts the stored record occurrence into the current occurrences of all sets in which it is defined with the option **INSERTION IS AUTOMATIC** (set S1 in this particular case);  
Note that the exact position of a new record occurrence within each set occurrence depends on the ORDER sub\_clause (**Order Is Last** in this particular case).
- 4. makes the new record occurrence be the current of run\_unit, the current of [record\_type] and the current of all CodasyI sets where it has been inserted and where it is defined as an owner;

### 7.3 INSERT Operator

If a member record type is declared with the option **INSERTION IS MANUAL**, member occurrences can be inserted into a set occurrence by a special INSERT operator.

Since the set S2 has been defined with the option **INSERTION IS MANUAL**, the stored R2 record occurrence does not become automatically a member of a S2 set occurrence.

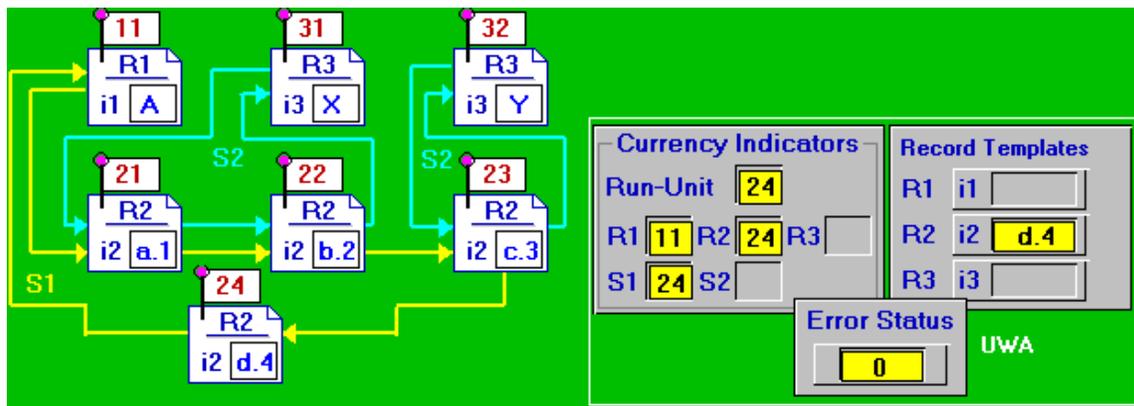
## Network (CODASYL) Data Model

To insert a record occurrence already stored in the database into the designated set occurrence we have to:

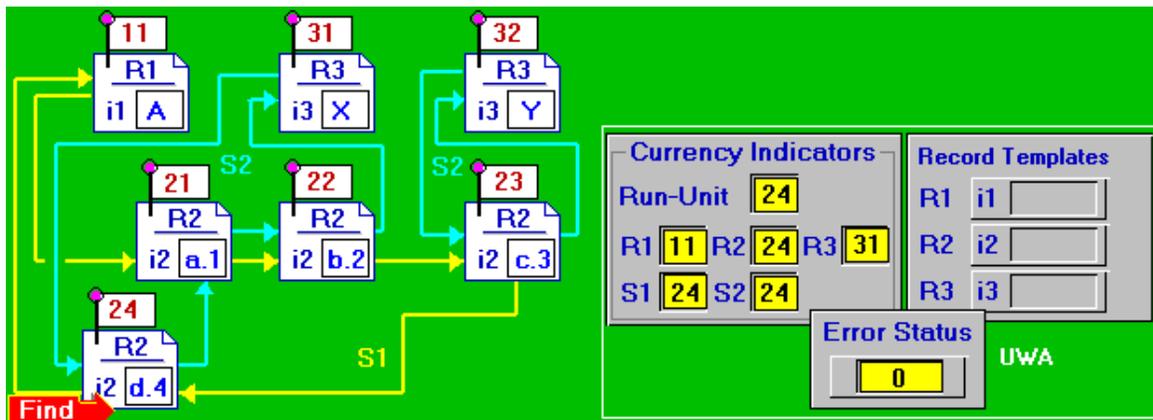
- 1. make this set occurrence the current of set type, by any suitable means;
- 2. make the record occurrence which is being inserted into the set occurrence, the current of run\_unit;
- 3. issue the command:

**Insert [record\_type] Record Into [set\_type] Set;**

For example,



x=31; Find R3 Record by Database key x;  
y=24; Find R2 Record by Database key y;  
Insert R2 Record Into S2 Set;



Note that the position of the inserted record occurrence within the [set\_type] set depends on the **ORDER** sub-clause (**ORDER IS FIRST**, in this particular case).

Note also that such an inserted record occurrence becomes the current of the [set\_type] set (S2 in this particular case).

Normally, the STORE and INSERT operators are combined into one application.

For Example,

**Get (R1.i1, R2.i2, R3.i3);**  
**Find R1 Record by CALC-key;**

Find R3 Record by CALC-key;  
 Store R2 Record;  
 Insert R2 Into S2 Set;

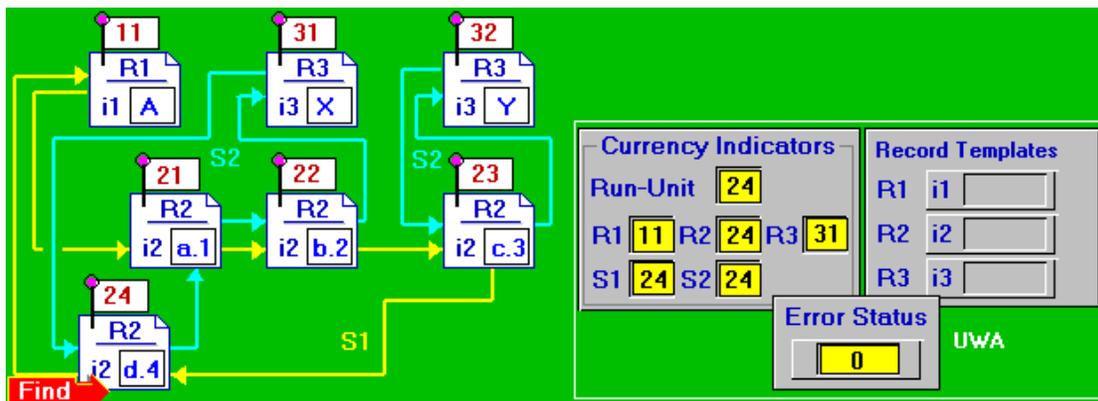
### 7.4 REMOVE Operator

We are permitted to execute the **REMOVE** statement if the option **RETENTION IS OPTIONAL** is specified for this set type.

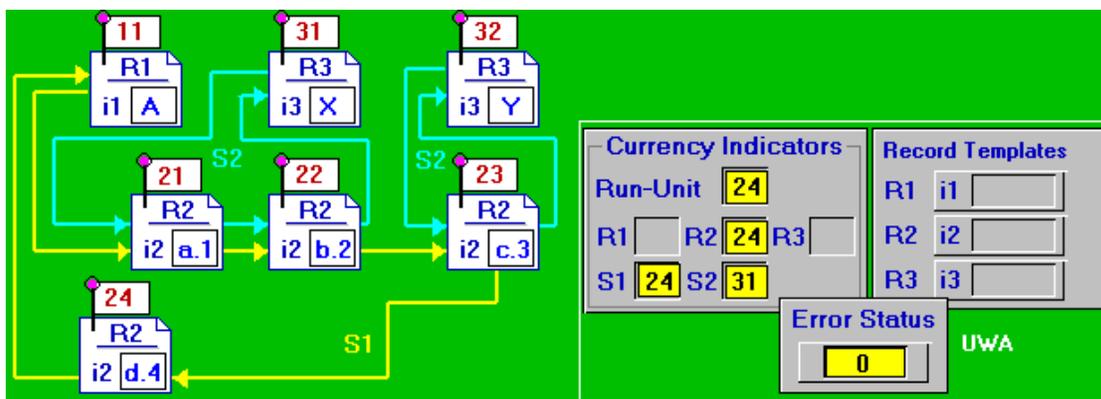
To remove a record occurrence of a certain type (say, R2) from a certain set occurrence (say, S2), we have to:

- 1. make designated record be the current of run\_unit;
- 2. issue the command:  
**Remove [record\_type] Record From [set\_type] Set;**

For example,



x=24; Find R2 Record by Database key x;  
 Remove R2 Record From S2 Set;



It should be noted that the record occurrence removed from a set occurrence is not deleted from the database. Thus, this record occurrence can be found as member of other sets, and/or it can be inserted into a designated set occurrence of the same type.

Attention ! The record occurrence removed from a set occurrence remains the current of run\_unit, current of its record type and current of all sets where it still participates as

owner or member (set S1 in this particular case).

A record occurrence prior to such removed record becomes the current of designated set type (see Currency Indicator S2).

Scanning a set occurrence in order to remove all the members is very common. As an example, consider the following application:

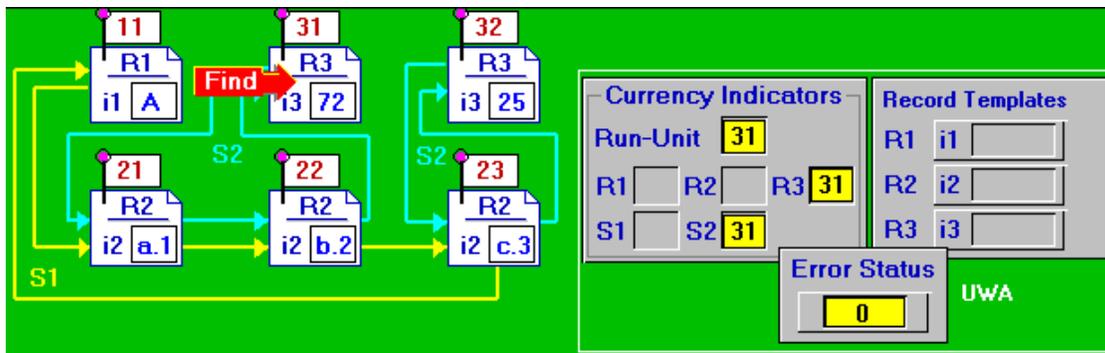
```

x=31; Find R3 Record by Database key x;
Loop: Find Next R2 Record In S2 Set
  If Error_Status=0 then begin;
    Remove R2 From S2 Set;
    Go to Loop;
  End;
  ...
  
```

### 7.5 MODIFY Operator

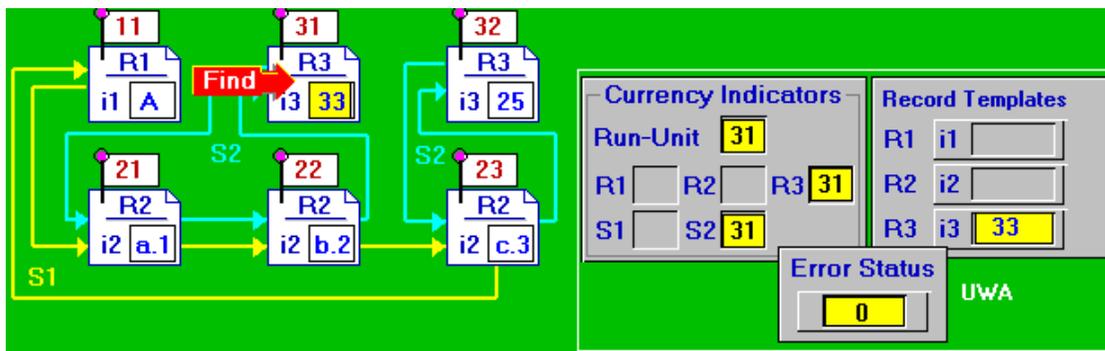
The **MODIFY** statement simply copies values from a record template into the current of run\_unit.

For example:



```

x=31; Find R3 Record by Database key x;
R3.i3="33";
Modify;
  
```



The **MODIFY** operator is frequently used after the **GET** command which reads the record occurrence from the database into its template.

For example:

```

x=31; Find R3 Record by Database key x;
  
```

Get R3 Record;  
R3.i3=R3.i3 + 5;  
Modify Record;

### 7.6 DELETE Operator

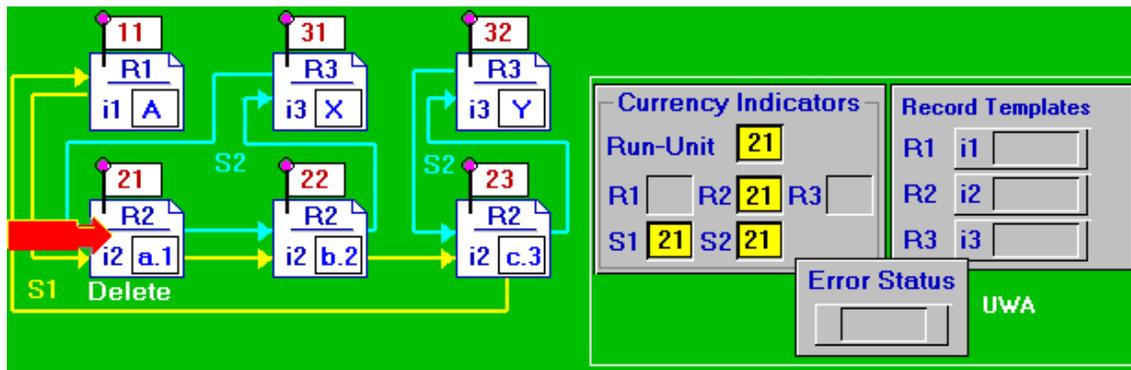
In fact, there exist many different forms of the DELETE statement.

Thus, for example, the command:

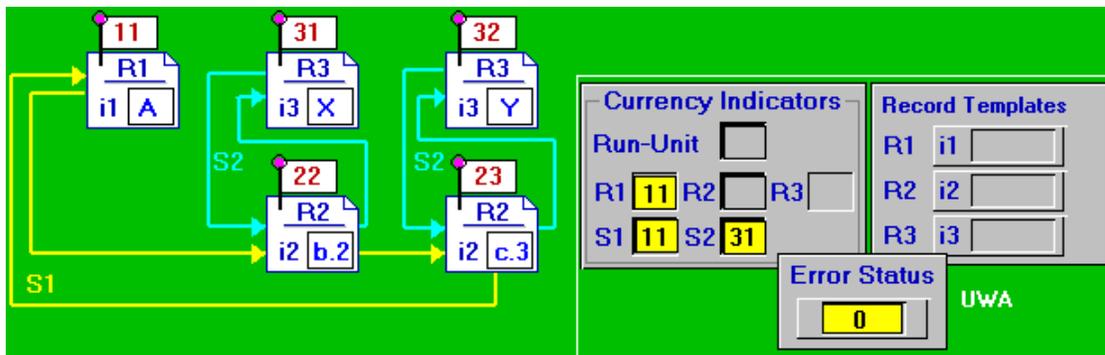
**DELETE [record\_type] RECORD;**

- 1. Deletes the current of from the database.
- 2. If the current of is a member of any set occurrence, is removed from those occurrences in a way similar to the REMOVE statement.
- 3. Deletes all references to such deleted record occurrence from UWA.

For example:



x=21; Find R2 Record by Database key x;  
Delete R2 Record;



If the deleted current of [record\_type] is the owner of some set occurrences, these occurrences must presently have no members (be "empty" occurrences), or an error is detected, and the deletion fails.

For example:

x=31; Find R3 Record by Database key x;  
Delete R3 Record;

## Network (CODASYL) Data Model

Thus, we need to scan a set occurrence in order to delete (or remove) all the members before deleting an owner.

For example,

```
x=31; Find R3 Record by Database key x;  
Loop: Find Next R2 Record In S2 Set;  
  If Error_Status=0 then begin;  
    Delete R2 Record; Go to Loop; End;  
  Delete R3 Record;
```

The **DELETE ALL** statement not only erases the current of run\_unit, as the simple **DELETE** statement does, but it is recursively applied to all members of the set occurrence owned by the deleted record.

For example,

```
x=31; Find R3 Record by Database key x;  
  Delete ALL;
```