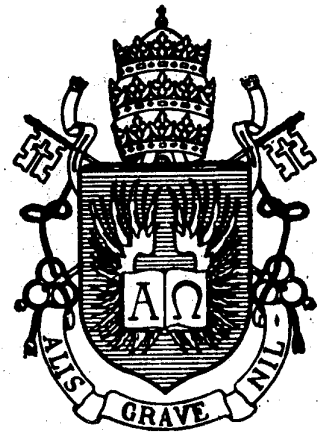


PUC

Series: Monografias em Ciência da Computação
Nº 17/78

JACKDAW II - Preliminary Specification

Part 2
Structural Discription

by

Michael F. Challis

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225 – CEP-22453
Rio de Janeiro – Brasil

Series: Monografias em Ciência da Computação No. 17/78
Series Editor: Michael F. Challis October, 1978

Jackdaw II - Preliminary Specification*

Part 2

Structural Description

by

Michael F. Challis

*This research has been sponsored by FINEP and CNPq.

Abstract

This monograph forms part of the preliminary definition of the Jackdaw II data base package (see the Preface).

It describes the facilities provided for the representation of data and data relationships in a data base, and the 'access paths' which may be followed to retrieve items stored within it.

Keywords: data base, data structures

Resumo

Esta monografia faz parte da definição preliminar do sistema de banco de dados Jackdaw II (veja o Prefácio).

São descritas as facilidades disponíveis para a representação de dados, e de relações entre dados, dentro de um banco de dados, e os 'caminhos de acesso' que podem ser seguidos para recuperar os itens armazenados.

Palavras chave: banco de dados, estruturas de dados

CONTENTS

Preface

This monograph forms part of the preliminary definition of the Jackdaw II data base package. The complete definition will include the following sections:

1. Introduction

   This section gives an overview of the package and introduces its major components.

2. Structural Description

   This describes the facilities provided for the representation of data and data relationships in a data base, and the 'access paths' which may be followed to retrieve items stored within it.

3. The Definition Language

   This contains a description of the language used to define the structure of a data base (often called the 'DDL'). The language also includes facilities for modifying the structure of an existing data base.

4. The Procedural Interface

   The 'nucleus' of the package will consist of a library of procedures which may be called to create, modify or delete items within a data base. This section describes these 'interface procedures'. The package will be written in BCPL, and the library of interface procedures will also initially be available only to BCPL application programs.

5. Concurrent Access

   The options available for data base access in both single-user and multi-user environments are described, including the facilities provided to help maintain data base consistency and integrity.

6. The SLIDE system

This section describes the language intended for the 'end-user'. SLIDE includes facilities for 'loading' (the addition of relatively large volumes of data) and report generation as well as facilities for casual interrogation and updating. The language has two levels - a 'low level', with commands which correspond approximately to the interface procedures of the nucleus, and a 'high level' with more powerful primitives.

The high level language may be easily extended to generate languages 'tailored' to particular applications.

Jackdaw II is a development of the Jackdaw data base package, which was originally developed by the author at the University of Cambridge, England (where it has been in production use since 1973), and which is now also available on the IBM 370/165 at PUC-RJ.

References

[1] - Richards, M., 1974. "The BCPL Programming Manual", Computer Laboratory, University of Cambridge, Cambridge, England.

[2] - Challis, M.F., 1974. "The JACKDAW Data Base Package", Proc. SEAS Spring Technical Meeting, St. Andrews, Scotland, April 1974. (Also available as "Technical Report No. 1" from the Computer Laboratory, University of Cambridge, Cambridge, England).

[3] - Challis, M.F., 1978. "Técnicas de Integridade no Sistema de Jackdaw de Bancos de Dados", to appear in the proceedings of "V Seminário Integrado de Software e Hardware Nacionais", Rio de Janeiro, Brasil. (Also available in English as "Integrity Techniques in the Jackdaw Database Package", Monografia em Ciência da Computação no. 9/77, Dept. de Informática, Pontifícia Universidade Católica, Rio de Janeiro, Brasil).

[4] - Challis, M.F., 1978. "Database Consistency and Integrity in a Multi-user Environment", to appear in Proc. International Conference on Data Bases: Improving Usability and Responsiveness, Israel, 1978 - Academic Press, New York. (An earlier version of this paper with the same title is available as Monografia em Ciência da Computação no. 6/78, Dept. de Informática, Pontifícia Universidade Católica, Rio de Janeiro, Brasil).

2.1 Entries and Classes

Information is stored in a data base in "entries", each of which must belong to some "class". (It may initially be helpful to think of an entry as a record, and a class as a set of records all of the same type, although the analogy is not a good one.)

Information within an entry is held in named "primitive fields", "group fields" and "link fields". Primitive fields contain (atomic) values, and group fields contain possibly multiple occurrences of groups of primitive fields. Link fields determine the relationships which exist between different entries in a data base.

The number, nature and names of the fields of an entry are determined by the class to which the entry belongs.

Example:

IDNUM, NAME and SALARY are primitive fields of the class EMPLOYEE.

CURRPROJS is a link field, which contains references to entries of another class (say PROJECT) which describe the tasks to which the employee is currently assigned.

SALARY_HISTORY is a group field composed of two primitive fields DATE and SALARY. An occurrence of this pair of fields is added to an EMPLOYEE entry whenever his salary is changed.

A particular EMPLOYEE entry might contain the following information:

```
EMPLOYEE:
  IDNUM = 176
  NAME = BURTHORPE, J.K.
  SALARY = 4200
  CURRPROJS = (P439, P476)
  SALARY_HISTORY:
    SALARY = 3400, DATE = 1/9/76
    SALARY = 3500, DATE = 1/1/77
    SALARY = 3900, DATE = 1/1/78
```

A class is either "keyed" or "keyless"; in the former case, one or more primitive fields of the class are designated as "key fields", whose contents determine a unique "key" for each entry which distinguishes it from all the other entries of that class. (See section 2.5 for more details of keys.)

New entries may be added to a class and existing entries deleted. If the class is keyed, a suitable key value must be supplied at the time of creation; this value may not be subsequently altered.

An entry in a keyed class may be accessed directly by quoting the name of the class and the key of the entry.

The entries of a class may be accessed sequentially, and, if the class is keyed, the entries will be supplied in ascending order of key value.

Example:

The IDNUM field is the key for the class EMPLOYEE. Each EMPLOYEE must have a different IDNUM, and a particular EMPLOYEE entry may be accessed directly by quoting this value:

e.g. 'EMPLOYEE 176'

Each class may have one or more named "indices" associated with it. An index, like a key, is defined by a sequence of one or more primitive fields of the class. Unlike key values, the index values of two different entries may be the same, and an index value may be altered at any time. (See section 2.6 for more details of indices.)

The entries of a class may be accessed sequentially with respect to an index, in which case the entries are supplied in ascending order of index value.

Example:

An index BYNAME is defined by the NAME field of the class EMPLOYEE. Sequential access to EMPLOYEE entries with respect to this index will supply the entries in alphabetical order of surname.

2.2 Primitive Fields

The definition of a primitive field includes a specification of the "basic type" of the field. The basic types available are:

BOOL - a 1-bit field
WORD - a 32-bit field
BYTE-n - a fixed-length n-byte field (1<=n<=256)
STRING - a variable-length field which may hold up to 255 bytes

Values of the appropriate size may be stored in and retrieved from primitive fields, although there are certain restrictions concerning the updating of key fields - see section 2.5.

Whenever a value is stored, the field is said to be "set"; it is possible to determine whether a particular field is set or not, and also to "unset" it.

Example:

HOLIDAYS is a primitive field of EMPLOYEE. If it is set, it contains the outstanding holiday entitlement due to the employee for the current year (which may be zero). If it is unset, it means that the employee is not entitled to holiday leave (perhaps he has recently joined the company, or is a consultant).

2.3 Group fields

A group field is defined as a named set of primitive and/or group fields. Each occurrence of a group field within an entry consists of one or more "group elements", or is "empty" (i.e. unset). Each group element contains one occurrence (possibly unset) of each of the primitive and group fields of the group.

Example:

The class EMPLOYEE includes a group field CHILDREN composed of primitive fields NAME and AGE and a further group field VACCINATION_HISTORY; this 'subgroup' field contains two primitive fields VACCIN and DATE. We can represent the structure of the group field CHILDREN as:

```
CHILDREN
  NAME
  AGE
  VACCINATION_HISTORY
    VACCIN
    DATE
```

A particular EMPLOYEE entry might include the following occurrence of the group field CHILDREN:

```
CHILDREN:
  NAME = DAVID, AGE = 5, VACCINATION_HISTORY:
                           VACCIN = POLIO1, DATE = 3/3/74
                           VACCIN = POLIO2, DATE = 8/9/74
  NAME = JOHN,  AGE = 0, VACCINATION_HISTORY:  (empty)
  NAME = SUSAN, AGE = 4, VACCINATION_HISTORY:
                           VACCIN = POLIO1, DATE = 4/4/75
```

This occurrence of CHILDREN contains three group elements. The first of these contains a two-element occurrence of VACCINATION_HISTORY, and the second contains an empty occurrence of this field.

Note that there is a difference between a group field occurrence which is unset, and an occurrence in which all the fields of each element are unset.

In many ways, the elements of a group field are analogous to the entries of a class; a group field may be keyed or keyless, and may possess one or more indices (see sections 2.5 and 2.6).

New elements of a group field may be created, and existing ones deleted; and facilities analogous to those provided for the entries of a class are available for direct and sequential access to the elements of a group field.

Example:

The key for the CHILDREN group is NAME, and an index BYAGE is defined by the primitive field AGE. As a consequence, each child is identified by its name, which must be supplied when the corresponding group element is created; the NAME field may not be subsequently altered.

VACCINATION_HISTORY is a keyless group with an index BYDATE composed of the primitive field DATE. Access to VACCINATION_HISTORY elements with respect to the index BYDATE provides vaccination information in historical order; a vaccination date may be corrected without deleting and recreating the element.

2.4 Link Fields

Each link field of an entry consists of a number (possibly none) of "link elements", each of which refers to an entry of some class; this class is known as the "partner class" of the link field. If a particular occurrence of a link field contains no link elements, it is said to be "empty", or "unset".

Example:

The link field CURRPROJS of the EMPLOYEE entry 176 in section 2.1 contains two link elements referring to entries of class PROJECT; PROJECT is the partner class of the link field CURRPROJS.

A fundamental property of the data base structure is that links between entries must always be two-way. This means that if an entry X1 contains a link element LE1 referring to an entry X2, then entry X2 will contain a corresponding link element LE2 referring to X1. LE2 is the "partner link element" of LE1, and vice-versa.

A corollary is that if a class C1 contains a link field L1 with partner class C2, then class C2 must contain a corresponding link field L2 with partner class C1. L2 is the "partner link field" of L1, and vice-versa.

Example:

The class PROJECT is a keyed class whose key is the primitive field PROJNUM. It includes the link field MEMBERS with partner class EMPLOYEE, which corresponds to the link field CURRPROJS in EMPLOYEE. MEMBERS and CURRPROJS are partner link fields.

EMPLOYEE 176 is attached to PROJECTs P439 and P476; suppose that EMPLOYEE 249 is also attached to PROJECT P439:

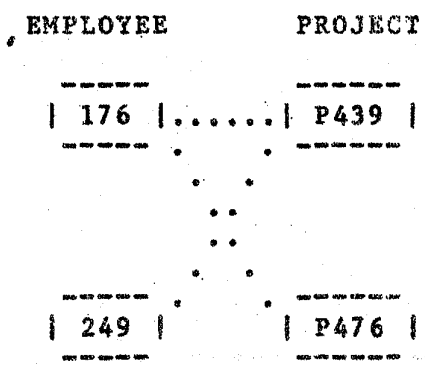
```
EMPLOYEE:
  IDNUM = 249
    ...
  CURRPROJS = (P439)
```

Then the link elements in the MEMBERS fields of the two PROJECT entries are as follows:

```
PROJECT:
  PROJNUM = P439
  MEMBERS = (176, 249)

PROJECT:
  PROJNUM = P476
  MEMBERS = (176)
```

We can illustrate the relationships between these EMPLOYEE and PROJECT entries as:



New elements may be added to a link field, and existing ones deleted; the data base routines automatically create or destroy the corresponding partner link elements to ensure that the two-way nature of links is maintained.

Example:

If we delete the link element in the MEMBERS field of PROJECT P439 that refers to EMPLOYEE 176, then the system will automatically remove the link element in the CURRPROJS field of EMPLOYEE 176 that refers to PROJECT P439.

A link field may be keyed or keyless; if it is keyed, the key is defined in terms of primitive fields of its partner class. It is often the case that the key for a link field is the same as the key of its partner class.

Whenever a new link element referring to an entry E is added to a keyed link field, the corresponding key fields in E are 'frozen': that is, they must not be altered until the link element is destroyed. The keys of all the link elements within a particular link field must be distinct.

Named indices may also be associated with a link field; each index is defined in terms of primitive fields of the partner class.

A link element in a keyed link field may be accessed directly, and sequential access to all the link elements of a link field is always possible. If the link field is keyed, the elements will be provided in ascending order of key; or, alternatively, sequential access with respect to an index may be requested in which case the elements will appear in order of index value. More details on link field keys and indices appear in sections 2.5 and 2.6.

We shall see in sections 2.4.1 and 2.4.2 that information may be associated with a link element in the form of marks and link element parameters; this information is directly accessible as soon as the link element has been located. It is also possible to 'follow' the link element to gain access to the entry to which it refers.

Examples:

i) The key for the link field CURRPROJS is the field PROJNUM of PROJECT; in a similar way, the key for MEMBERS is the same as the key of its partner class EMPLOYEE. In this case, the user need take no special precautions with respect to key fields when adding link elements, since the link field key values are guaranteed distinct because they are also key values for the classes.

An index BYSALARY is defined for the link field MEMBERS, and consists of the SALARY field in the class EMPLOYEE. Sequential access with respect to BYSALARY supplies the link elements referring to those employees currently attached to the project in ascending order of salary. Each link element may be 'followed' to access the EMPLOYEE entries themselves.

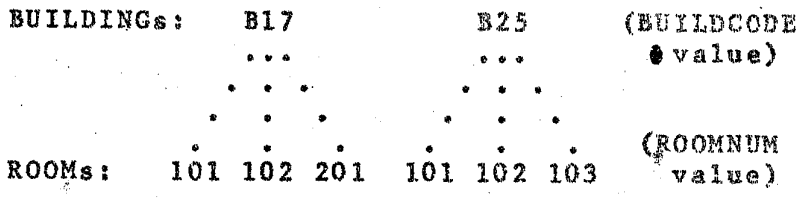
ii) Two classes BUILDING and ROOM are defined. BUILDING is a keyed class with key BUILDCODE and ROOM is a keyless class with a primitive field ROOMNUM. We wish to link each room to the building of which it is a part; we know that room numbers are unique within a building, although two rooms in different buildings may have the same number.

The two partner link fields are called ROOMS (in BUILDING) and BUILDING (in ROOM); the key for ROOMS is ROOMNUM and for BUILDING is BUILDCODE.

Before requesting the creation of a link between a BUILDING B and a ROOM R, we must ensure that the ROOMNUM field in R is set; and the system will reject the request if the ROOMS link field of B already includes a link element referring to a ROOM with the same room number.

The ROOMNUM field of a ROOM R may only be altered if its BUILDING field is empty; for if its BUILDING field contains a link element referring to a BUILDING B, then B's ROOMS field must contain the partner link element PL referring to R, and the room number is PL's key.

The relationships between some buildings and their rooms might be as follows:



Link fields are of two kinds: "uni-link fields", which contain a maximum of one link element, and "multi-link fields" which may contain any number of link elements.

Example:

The ROOMS link field of a BUILDING is defined as a multi-link field; on the other hand, the BUILDING link field of a ROOM is defined as a uni-link field, since each room can belong to one building only.

One class may contain two or more link fields with the same partner class; in other words, it is possible to represent more than one relationship between two classes at the same time.

Example:

The CURRPROJS, MEMBERS link fields represent the employees working on a particular project on a day-to-day basis. Each project also has an auditor whose job it is to monitor progress from time to time. The two uni-link fields AUDITPROJ, AUDITOR (in classes EMPLOYEE and PROJECT respectively) define this second relationship between employees and projects.

It is also possible to define one or more relationships amongst the entries of one class - that is, the partner class of a link field may be the same as its class.

Example:

Link fields MANAGER and EMPLS are defined for the class EMPLOYEE; MANAGER is a uni-link field and EMPLS is a multi-link field. The key for both is the IDNUM field. The MANAGER link field for each employee (except the chairman of the company) contains a link element referring to the employee's manager. The EMPLS link field is only set for those employees who are managers, and contains link elements referring to each of the employees directly responsible to that manager.

No link field may include two link elements referring to the same entry.

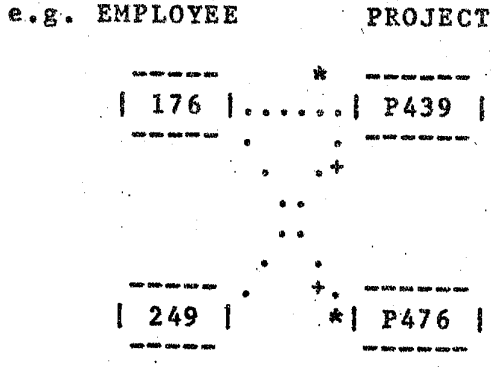
2.4.1 Marks

It is possible to select one (but not more) of the link elements within a link field and "mark" it; the partner link element is then said to be "chosen". A marked link element automatically becomes unmarked when a different link element in the same field is marked; it is also possible to "unmark" a link element explicitly, thus leaving the link field without any marked element.

Marks are named, and several different marks may be associated with one link field of a class.

Example:

The link field MEMBERS of the class PROJECT has mark fields LEADER and TEABOY. In any particular PROJECT entry, just one of the link elements in the MEMBERS field may be marked as the LEADER, and just one as TEABOY.



* - indicates a LEADER-marked link element
\+ - indicates a TEABOY-marked link element

In this example, EMPLOYEE 176 is the project leader for both projects; he also makes the tea for PROJECT P476, since there is no-one else to do it. EMPLOYEE 249 is the tea boy for PROJECT P439. Both link elements in the CURRPROJS field of EMPLOYEE 176 are chosen with respect to the LEADER mark, but only the second is chosen with respect to the TEABOY mark.

Given a particular link element, it is possible to determine whether it is marked with respect to a particular mark of the link field, or whether it is chosen with respect to a particular mark of the partner link field.

A marked link element within a link field may be accessed directly, and the chosen elements may be accessed sequentially.

Example:

The link element referring to the leader of a project may be accessed directly from the MEMBERS link field.

The link elements referring to those projects for which a given employee makes the tea may be accessed sequentially from his CURRPROJS field.

2.4.2 Link element parameters

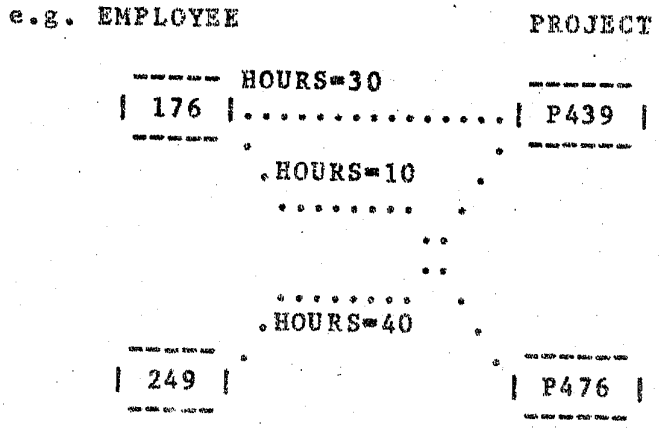
The definition of a link field may include the definition of one or more primitive and/or group fields which are to form part of every link element in that field. These primitive and group fields are called "link element parameters".

Once a link element has been located, its parameter fields may be accessed in just the same way as the primitive and group fields of an entry may be accessed.

Link element parameters may be used to store information which is a function of the relationship between two entries, rather than a function of one entry or the other.

Example:

The field CURRPROJS of the class EMPLOYEE has as link element parameter a primitive field HOURS. The HOURS field of a link element in an EMPLOYEE E referring to a PROJECT P contains the number of hours per week which E devotes to the project P.



Here EMPLOYEE 176 devotes 30 hours a week to PROJECT P439 and 10 hours a week to PROJECT P476.

Note that the hours worked cannot be stored either as a primitive field of the EMPLOYEE entry or of the PROJECT entry - they are genuinely a function of the combination of the two.

2.5 Keys

We have seen above that a class (or group or link field) may be keyed in order to provide direct access to the entries (or group or link elements) within it.

In all three cases, a key is defined as a sequence of one or more primitive fields p1, p2, ... pn. These are known as "key fields"; p1 is the "primary key field" and p2, ... pn are the "secondary key fields".

The key of an entry (or group or link element) is defined as the sequence of values of those of its key fields that are set. The primary key field must always be set, but secondary key fields may remain unset subject to the condition that if pi is unset, then so are p(i+1), p(i+2), ... pn.

The key fields in a class key must be primitive fields of the class at the 'outer level' - that is, they must not be link element parameters, or part of a group field of that class. The key of an entry must be supplied when the entry is created, and may not be subsequently updated. All the entries in one class must have different keys.

The key fields in a group field key must be primitive fields belonging to that group, but not to any other group field contained within it. The key of a group element must be

supplied when the element is created, and may not be subsequently altered; all the group elements in a particular occurrence of a group field must have different keys.

The key fields in a link field key must be 'outer-level' primitive fields of the partner class of the link field. When a new link element referring to an entry E is created, the values of the corresponding key fields in E must define a key different from that of any other link element belonging to the same link field; and these link key fields in E may not be updated as long as the link element continues to exist.

Sequential access may be requested to all entries (or group or link elements) of a class (or group or link field) whose key is greater than some given value; in this case the items are supplied in ascending order of key.

Comparison of two keys V = v1, v2, ... vp and W = w1, w2, ... wq (where p,q<=n, the number of key fields) is defined as follows:

i) V = W iff p=q and vi=wi for all 1<=i<=p

ii) V < W iff
v1=w1, v2=w2, ... v(k-1)=w(k-1) and vk<wk
for some k<=min(p,q),
or vi=wi for all 1<=i<=p and p<q

(In other words, keys are compared from the left to the right, and any unset fields at the end are considered to be less than the value of any set field.)

The ordering of field values may be defined by the user (see section 2.7); if not, the following defaults are used according to the basic type of the field:

BOOL: FALSE < TRUE

WORD: The 32-bit value is interpreted as a signed, 2's complement integer.

BYTE-n: Logical comparison - the 8n-bit value is interpreted as a positive integer.

STRING: Lexicographic comparison - if S and T are two strings composed of bytes s1, s2, ... sn and t1, t2, ... tm respectively, then S < T if:
s1=t1, s2=t2, ... s(i-1)=t(i-1) and si<ti
for some 1<=i<=min(n,m),
or n<m and si=ti for all 1<=i<=n

Example:

A class PERSON is defined with primitive fields SURNAME and CHRISTIAN_NAMES of type STRING, and DISCRIMINATOR of type BYTE-1. The key for the class is defined to be SURNAME, CHRISTIAN_NAMES, DISCRIMINATOR. The field DISCRIMINATOR is only set if there are two or more people with identical surnames and christian names.

2.6 Indices

One or more named indices may be associated with a class (or group or link field), irrespective of whether it is keyed or not.

An index is defined as a sequence of one or more primitive fields p1, p2 ... pn called "index fields". The index value of an entry (or group or link element) is defined as the sequence of values of its index fields, where the symbol '*' is used to indicate the 'value' of any unset field. (Note that unlike keys, any or all of the index fields may be unset.)

The rules governing the choice of index fields for a class (or group or link field) index are the same as those given in section 2.5 governing the choice of key fields. However, index values need not be provided when an entry (or group or link element) is created, they do not need to be unique, and may be changed at any time.

Sequential access with respect to an index may be requested to all entries (or group or link elements) of a class (or group or link field) whose index value is greater than some given value; in this case, the items are supplied in ascending order of index value.

Comparison of two index values V = v1, v2, ... vn and W = w1, w2, ... wn (where n is the number of index fields) is defined as follows:

i) V = W iff vi=wi for all 1<=i<=n

ii) V < W iff
v1=w1, v2=w2, ... v(k-1)=w(k-1) and vk<wk
for some k<=n

where '*' (the unset value) is always less than any possible real value.

The ordering of field values may be defined by the user (see section 2.7); if not, the defaults described in section 2.5 are used.

Example:

A class WORKMAN is defined with STRING primitive fields SURNAME and CHRISTIAN_NAMES and WORD primitive field AGE.

An index BYNAME is defined as SURNAME, CHRISTIAN_NAMES. It does not matter if two workmen happen to have the same names.

Another index BYAGE is defined as AGE, SURNAME. Sequential access via BYAGE will supply workmen in ascending order of age; workmen of the same age will be in alphabetical order of surname. Those workmen who refuse to reveal their age (i.e. whose AGE field is unset) will appear first.

2.7 Comparison functions

A user-defined "comparison function" may be associated with any key (or index) field which forms part of the key (or index) of a class, group or link field. Each comparison function is identified by its (unique) name.

The nucleus will make calls 'R := CMP(V, W)' of a comparison function CMP as necessary; the result R is interpreted as follows:

R = 0	means	V = W
R < 0	means	V < W
R > 0	means	V > W

Note that a comparison function is not associated with a primitive field, but with a primitive field in its capacity as a component of a particular key or index.

Example:

Two separate indices are defined for the class EMPLOYEE; UPSALARY and DOWNSALARY. Both are defined by the index field SALARY. In the DOWNSALARY index only, a comparison function F is associated with the index field SALARY, where:

F(V, W) = 0	if	V = W
= -1	if	W < V
= +1	if	W > V

2.8 Classes with variants

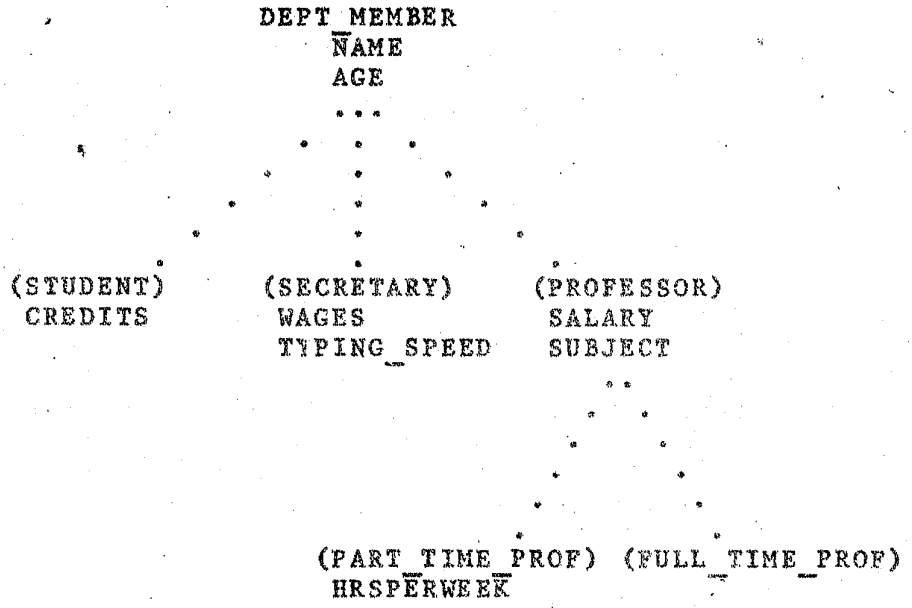
It is often desirable to consider a set of entries of similar (but not identical) structure as all belonging to a single class. This may be done by defining a class with "variants". The fields common to all the entries are defined as part of the class proper, and the fields which belong only to some subset of the entries are defined as part of one of the variants.

Each variant may itself have further variants; if it does not, it is called a "basic variant". The variants out of which a class (or variant) is immediately composed (i.e. which belong to that class (or variant) but not to any variant contained within it) are called the "direct variants" of that class (or variant).

Example:

DEPT_MEMBER is a class whose entries represent the members of a university department, and the class has three direct variants STUDENT, SECRETARY and PROFESSOR corresponding to the three different categories of people in the department.

The variant PROFESSOR itself contains two further direct variants PART_TIME_PROF and FULL_TIME_PROF. Primitive fields are associated with the class and each of its variants as shown in the following diagram:



The variants STUDENT, SECRETARY, PART_TIME_PROF and FULL_TIME_PROF are the basic variants of the class DEPT_MEMBER.

Each entry of a class with variants must have a structure corresponding to one of the basic variants, and, given a particular entry, it is possible to determine to which variant(s) it belongs.

Example:

The structure of an entry of the class DEPT_MEMBER must correspond to that of a STUDENT, SECRETARY, PART_TIME_PROF or FULL_TIME_PROF. Here are some examples of DEPT_MEMBER entries:

```
DEPT_MEMBER (STUDENT):
  NAME = JOHN
  AGE = 19
  CREDITS = 20

DEPT_MEMBER (SECRETARY):
  NAME = SUSAN
  AGE = 24
  WAGES = 60
  TYPING_SPEED = 100

DEPT_MEMBER (PART_TIME_PROF):
  NAME = SEAN
  AGE = 30
  SALARY = 5000
  SUBJECT = DATABASE
  HRSPERWEEK = 20
```

Given the last of these DEPT_MEMBER entries (SEAN), we may determine first that it belongs to the direct variant PROFESSOR of DEPT_MEMBER and then that it belongs to the (basic) variant PART_TIME_PROF of PROFESSOR.

Suppose an entry E of class C belongs to a basic variant Vn where V(i+1) is a direct variant of Vi for $0 \leq i \leq n$ and V0 is the class C. Then while we know only that E belongs to the variant Vk, we may only access those fields defined in variants Vi for $0 \leq i \leq k$. These are called the fields "visible" in E with respect to the variant Vk. (If k=0, they are called the fields "visible" with respect to the class C.)

Example:

Suppose we are accessing DEPT_MEMBER entries sequentially, and arrive at the entry E for SEAN. At this stage we do not know to which variant E belongs, and so only the fields NAME and AGE are visible. After discovering that E belongs to the direct variant PROFESSOR, we may access the fields SALARY and SUBJECT; and when we finally ascertain the basic variant PART_TIME_PROF to which E belongs, we gain access to the HRSPERWEEK field, and so all the fields of E are visible.

The examples so far have shown only primitive fields associated with variants; but group fields and link fields may also be restricted to particular variants of a class; in other words, the partner class of a link field may be a variant.

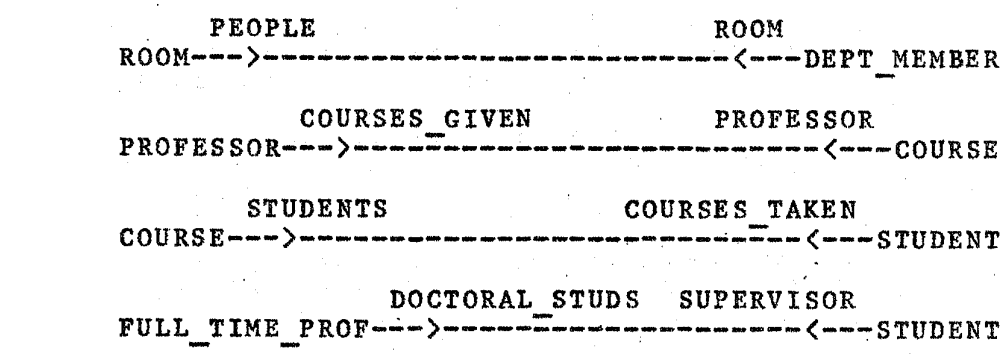
Examples:

i) A relationship between ROOM entries and DEPT_MEMBER entries shows in which room a particular person is located. The uni-link field ROOM belongs to the class DEPT_MEMBER as a whole.

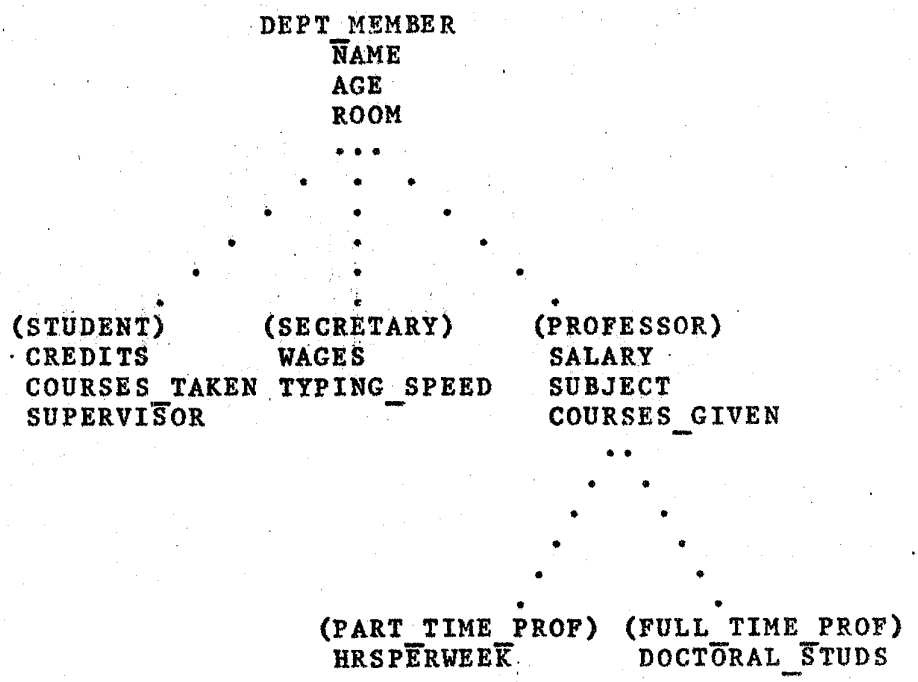
ii) A class COURSE contains entries describing the courses given by the professors in the department. The multi-link field COURSES_GIVEN belonging to the variant PROFESSOR indicates the courses offered by the professor, and the multi-link field COURSES_TAKEN belonging to the basic variant STUDENT indicates the courses which the student is attending.

iii) Only full-time professors may act as supervisors for doctoral theses, and a relationship between the basic variants STUDENT and FULL_TIME_PROF shows which students are supervised by whom.

The following diagram illustrates these relationships and includes the names of all the link fields:



The diagram showing which fields belong to which variants of the class DEPT_MEMBER is now:



Given a particular COURSE entry, we may sequentially access each of the STUDENTS link elements. If we follow one of these, we arrive at a DEPT_MEMBER entry which is already known to belong to the STUDENT variant, and so we immediately have access to all its fields including the SUPERVISOR uni-link field. Following the single link element in this field gives us access to another DEPT_MEMBER entry, known to belong to the basic variant FULL_TIME_PROF.

The key fields that define the key for a class with variants must be chosen from the primitive fields visible with respect to the class.

The key fields that define the key field for a link field whose partner class is a variant (or a class with variants) must be chosen from the primitive fields visible with respect to that variant (or class).

Similar remarks apply to index fields.

Examples:

i) Only the NAME and AGE fields may take part in the definition of the key for the class DEPT_MEMBER or the key for the link field PEOPLE in the class ROOM.

ii) Any of the fields NAME, AGE and CREDITS may take part in indices defined for the link field DOCTORAL_STUDS.

2.9 Format of names

The names of classes, fields, indices etc. may contain letters, digits, periods (.) and underlines (_), but must start with a letter.

Names must be chosen in such a way that each item is uniquely specified by its "fully qualified name", which is defined as a sequence of names as follows:

Let c be a class with name C and fully qualified name C'. (v,V,V'), (p,P,P'), (l,L,L'), (g,G,G'), (m,M,M') and (i,I,I') have similar meanings for variants, primitive fields, link fields, group fields, mark fields and indices respectively. Then:

i) C' = C

ii) V' = C', V
L' = C', L where c is the class in which the variant v or link field l is defined

iii) I' = C', I for an index i of class c

iv) P' = C', P
G' = C', G for fields p and g which belong to class c but are not embedded in any group or link field of c

v) M' = C', L', M for a mark field m in link field l of class c

vi) I' = C', L', I for an index i of link field l of class c

vii) P' = C', L', P
G' = C', L', G for fields p and g which belong to link field l of class c but are not embedded in any group field of l

viii) I' = G', I for an index i of group g

ix) P' = H', P
G' = H', G for fields p and g which belong to group field h, but are not embedded in any group field of h

Example:

Suppose a class c has two variants v and w. Primitive field p, link field l and group field g are common to both variants, and pv, pw are primitive fields local to variants v, w respectively. pl is a parameter of l, and pg is a primitive field in the group g.

The corresponding fully qualified names are:

```
C' = C
V' = C, V;   W' = C, W;   L' = C, L
P' = C, P;   G' = C, G;   PV' = C, PV;  PW' = C, PW
PL' = C, L, PL
PG' = C, G, PG
```

As a consequence, we see that:

i) V, W, L, P, G, PV and PW must all be different.

ii) C, L, PL and PG may all be the same.

2.10 The class definitions

The description of the structure of the entries in a data base is composed of a sequence of "class definitions". Each class definition describes the structure of a particular class, and includes "variant definitions", "primitive field definitions", "group field definitions" and "link field definitions" as required. In a similar way, each link field definition includes primitive field definitions to describe its parameters, and "mark field definitions" to describe its mark fields; and each group field definition contains primitive and group field definitions describing the fields contained within it. "key definitions" and "index definitions" may occur inside class, group field or link field definitions, and describe the keys and indices defined for these items.

The class definitions themselves form part of the data base, and may be interrogated.

Examples:

A program may determine:

- the basic type of a particular primitive field.
- the names of all the classes in a data base.

- the names of the primitive fields within a particular group field,
- the name of the partner field of a particular link field,
- the name of the class to which a particular link field belongs,
- whether a particular primitive field is a key or index field, and, if so, to which class, group or link field the key belongs.

By use of these interrogation facilities, it is possible to develop general purpose programs which are able to interact with any data base; the SLIDE system is an example of such a program.

To assist the development of higher-level systems based on Jackdaw, certain fields are provided in the class definitions in which "user information" pertinent to such systems may be stored. The nucleus enables a user to interrogate the user information fields, but does not itself take any notice of their contents. For convenience, the user information fields are classified as type, occurrence, description and miscellaneous fields.

"Implementation information", which, for example, may determine the storage technique used for a particular field, is also included in the class definitions, and may be accessed by applications programs.

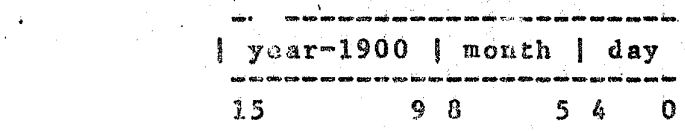
2.10.1 User type information

The "user type" field, associated with each primitive field definition, is, essentially, a refinement of the basic type of the field. A higher-level system (such as SLIDE) may access both the user type and the basic type of a primitive field and thus provide suitable checks and format conversions between the external and internal representations of values.

The user types available will depend on the system installation, but will always include the ones in the left hand column below; the right hand column lists examples of basic types compatible with the given user type:

User type	Basic types
BOOL	BOOL
STRING (variable length)	STRING, WORD, BYTE-n
INT	WORD, BYTE-n (n<=4)
REAL	BYTE-8
DATE	BYTE-2
CHAR-n (fixed length)	BYTE-n, WORD (if n=4), STRING

The conventional external and corresponding internal representations are defined in the SLIDE manual; for example, the internal representation of a DATE is:



Example:

The DATE primitive field in the SALARY_HISTORY group field of the class EMPLOYEE is defined as user type DATE, basic type BYTE-2.

Access to this field using SLIDE ensures that its content always represents a valid date, but access by means of the interface procedures READBYTES and SETBYTES of the nucleus will permit the retrieval or storage of any 16-bit value.

2.10.2 User occurrence information

"Occurrence information" may be associated with a primitive, group, link or mark field to say whether the field is "optional" or "mandatory". The intended meaning is as follows:

i) Each mandatory primitive field in an entry should be set; optional ones may be unset.

ii) Each mandatory link or group field within an entry should contain at least one element; optional ones may be empty.

iii) Suppose l is a link field which includes a mandatory mark field m. Then every non-empty link field l of an entry E must include an element marked m.

Example:

The primitive fields SALARY and DATE of the group field SALARY_HISTORY of EMPLOYEE are mandatory, but the group field itself is not.

2.10.3 User description information

Comments may be included within the class definitions in "user description" fields, which may be associated with class, variant, primitive field, group field, link field or mark field definitions. A user description field may hold a string up to 255 characters long.

Example:

The user description field associated with the primitive field definition for the field NAME of the class EMPLOYEE contains the comment: 'Surname first'.

2.10.4 User miscellaneous information

Finally, there is a "user miscellaneous" field associated with each class, variant, primitive field, group field, link field and mark field definition in which the user may place arbitrary data. The data is stored as a variable-length vector of 32-bit values, and no conventional interpretation is suggested.

Example:

A particular higher-level application might choose to store the codes and scrambled passwords of those users allowed access to restricted fields in the user miscellaneous fields of the relevant definitions.

2.10.5 Implementation information

Associated with each primitive, link and group field definition is an "implementation information" field in which the user may store an indication of the probable frequency of occurrence of the field. Two options are available: "rare" and "common".

Most occurrences of a rare primitive field are expected to be unset, and most occurrences of a rare group or link field are expected to contain no elements; the converse is true for common fields.

The storage allocation strategy adopted by the package for rare fields attempts to minimise wasted space at the expense of a slight increase in access time for rare fields.

Example:

Most employees stay with the company for a short time only, and so have no salary history; so the group field SALARY_HISTORY is designated as rare. But all employees have a name, and so the NAME primitive field is designated common.

2.11 Physical representation information

The techniques used for representing the information of a data base in a physical file are such that certain items of information are likely to be "close" together (i.e. probably in the same physical block of the disc), whereas others are likely to be "distant" (probably in different blocks).

Two entries of a class (or two elements of a group) with adjacent keys are likely to be close, and so sequential access by key is likely to be physically as well as logically efficient; on the other hand, entries or elements adjacent with respect to an index may be physically distant.

The primitive, group and link fields of an entry are close to the entry, and group (or link) elements are close to the group (or link) field. Link element parameters and marks and chosen information are close to the link element.

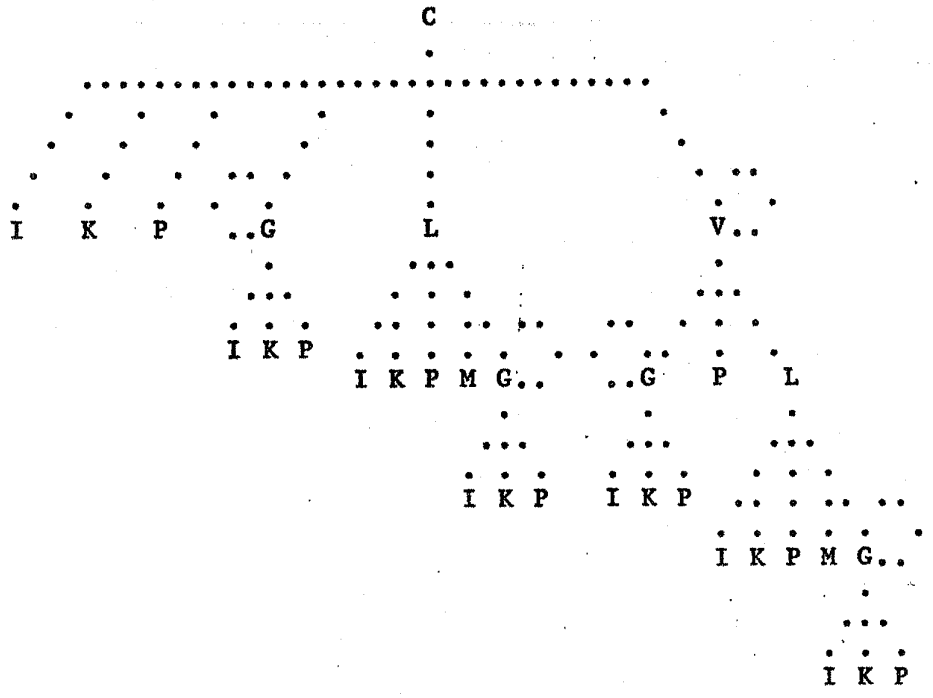
The key for a particular link element is held in primitive fields of the partner entry, and so is likely to be distant; so sequential access to the keys of the link elements within a link field may be a physically expensive process. (If this proves to be a common requirement, the key can be duplicated in link element parameters.)

The storage allocation strategy adopted by the package is designed to be efficient when a data base is in a 'steady state' - that is, when additions and deletions are random and roughly equivalent, so that the rate of change of overall size is small and evenly distributed over the data base. In such a situation, the allocation, release, and subsequent 'garbage collection' of unused space in the data file takes place automatically without the need for user intervention.

However, if large quantities of new data are to be added ('loading'), or a substantial number of deletions are to be made, special modes of operation are available where the internal automatic algorithms may be suspended.

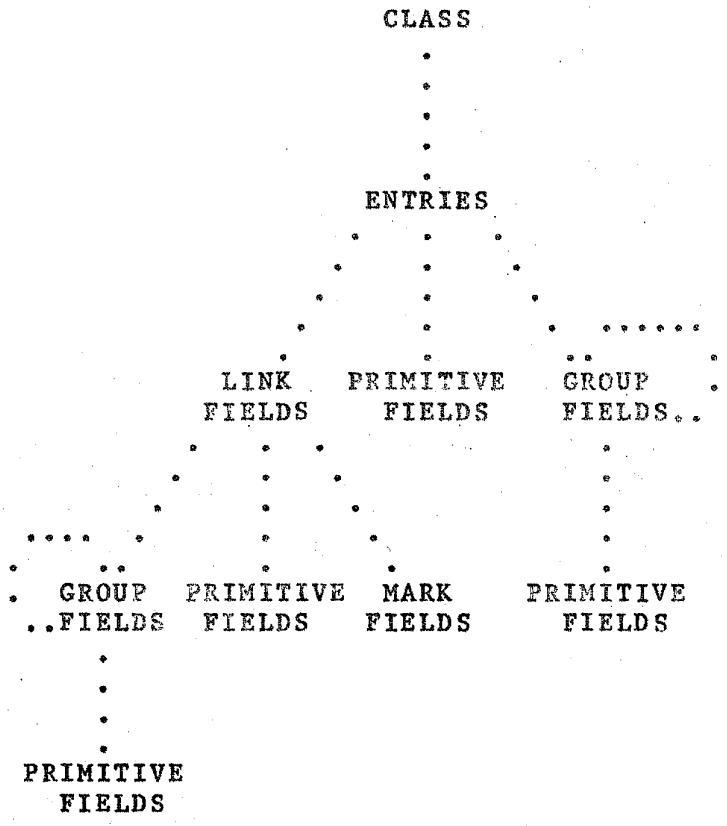
2.12 Summary diagrams

The following diagrams may be helpful as references; the first illustrates the relationships between the components of a data base definition, and the second illustrates the relationships between the components of the data itself.



where C - is a class definition
K - is a key definition
I - is an index definition
P - is a primitive field definition
G - is a group field definition
L - is a link field definition
M - is a mark field definition
V - is a variant definition

Data base definition relationships



Data base component relationships

2.13 Class definitions for the examples

The following program in the data base definition language (defined in chapter 3) defines the classes used in the examples of this chapter.

```
NEW MODULE CMPFS ( REVINT )

NEW CLASS EMPLOYEE
BEGIN

  DESCRIPTION IS
      "Each entry corresponds to a full-time employee"

//primitive fields
  INT IDNUM (MAND),
      SALARY (MAND),
      HOLIDAYS
  STRING NAME (MAND; DESCRIPTION IS "Surname first")
```

```
//group fields
  GROUP SALARY_HISTORY
  BEGIN
    RARE
    INT SALARY (MAND)
    DATE DATE (MAND)
    NOKEY
  END

  GROUP CHILDREN
  BEGIN
    STRING NAME (MAND)
    INT (BYTE-1) AGE
    GROUP VACCINATION_HISTORY
    BEGIN
      STRING VACCIN
      DATE DATE
      INDEX BYDATE IS DATE
    END
    KEY IS NAME
    INDEX BYAGE IS AGE
  END

//key and index information
  KEY IS IDNUM
  INDICES UPSALARY IS SALARY,
          DOWNSALARY IS SALARY (REVINT)

END

NEW CLASS PROJECT
BEGIN
  STRING (BYTE-4) PROJNUM
  KEY IS PROJNUM
END

NEW LINK ( AUDITPROJ (SINGLE, RARE),
           AUDITOR (SINGLE, MAND) )
             BETWEEN EMPLOYEE AND PROJECT

NEW LINK ( MANAGER (SINGLE),
           EMPLS (RARE) )
             BETWEEN EMPLOYEE AND EMPLOYEE

NEW LINK ( CURRPROJS (INT (BYTE-1) HOURS),
           MEMBERS (MARKS LEADER (MAND), TEABOY
                    INDEX BYSALARY IS SALARY) )
             BETWEEN EMPLOYEE AND PROJECT
```

```
NEW CLASS BUILDING
BEGIN
  STRING (BYTE-6) BUILDCODE (MAND)
  KEY IS BUILDCODE
END

NEW CLASS ROOM INT (BYTE-2) ROOMNUM (MAND)

NEW LINK ( ROOMS (KEY IS ROOMNUM),
           BUILDING (SINGLE, MAND; KEY IS BUILDCODE) )
             BETWEEN BUILDING AND ROOM

NEW CLASS PERSON
BEGIN
  STRING SURNAME (MAND), CHRISTIAN_NAMES
  CHAR-1 DISCRIMINATOR
  KEY IS (SURNAME, CHRISTIAN_NAMES, DISCRIMINATOR)
END

NEW CLASS WORKMAN
BEGIN
  STRING SURNAME, CHRISTIAN_NAMES
  INT (BYTE-1) AGE
  INDICES BYNAME IS (SURNAME, CHRISTIAN_NAMES),
          BYAGE  IS (AGE, SURNAME)
END

NEW CLASS COURSE
BEGIN
  INT COURSENUM (MAND)
  STRING TITLE
  KEY IS COURSENUM
END
```

```
NEW CLASS DEPT_MEMBER
BEGIN

  STRING NAME (MAND)
  INT (BYTE-1) AGE

  VARIANTS STUDENT (INT (BYTE-1) CREDITS),

           SECRETARY ( INT (BYTE-2) WAGES,
                           (BYTE-1) TYPING_SPEED ),

           PROFESSOR
           BEGIN
             INT SALARY
             STRING SUBJECT
             VARIANTS PART_TIME_PROF
                         (INT (BYTE-1) HRSPERWEEK),
                      FULL_TIME_PROF
           END

END

NEW LINK ( PEOPLE,
           ROOM (SINGLE) )
             BETWEEN ROOM AND DEPT_MEMBER

NEW LINK ( COURSES_GIVEN,
           PROFESSOR (SINGLE) )
             BETWEEN PROFESSOR OF DEPT_MEMBER
                AND COURSE

NEW LINK ( COURSES_TAKEN,
           STUDENTS )
             BETWEEN STUDENT OF DEPT_MEMBER AND COURSE

NEW LINK ( DOCTORAL_STUDS,
           SUPERVISOR (SINGLE) )
             BETWEEN STUDENT OF DEPT_MEMBER
                AND FULL_TIME_PROF OF
                      PROFESSOR OF DEPT_MEMBER
```