The Network Data Model

The network data model is the entity-relationship model with all relationships restricted to be binary, many-one relationships. This restriction allows us to use a simple directed graph model for data. In place of entity sets, the network data model tasks of logical record types. A logical record type is a name for a set of records, which are called logical records. Logical records are composed of fields, which are places in which elementary values such as integers and character strings can be placed. The set of names for the fields and their types constitute the logical record format.

Record Identity

There is a close analogy between these terms for networks and for relations, under the correspondence

Logical record format	Relation scheme
Logical record type	Relation name
Logical record	Tuple
Field	Attribute

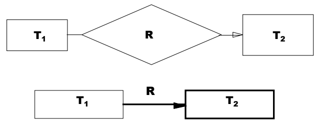
However, there is an important distinction between tuples of relations and records of a record type. In the relational model, tuples are the values of their components. Two tuples with the same values for the same attributes are the same tuple. The network data model is object-oriented, at least to the extent that it supports object identity.

Records of the network model may be viewed as having an invisible key, which is in essence the address of the record, i.e. its object identity. This unique identifier serves to make records distinct, even if they have the same values in their corresponding fields. In fact, it is feasible to have record types with no fields at all.

The reason it makes sense to treat records as having unique identifiers, independent of their field values, is that physically, records contain more data than just the values in their fields. In a database built on the network model they are given physical pointers to other records that represent the relationships in which their record type is involved. These pointers can make two records with the same field values different, and we could not make this distinction if we thought only of the values in their fields.

Links

Instead of binary many-one relationships we talk about links in the network model. We draw a directed graph, called a network, which is really a simplified entity-relationship diagram, to represent record types and their links. Nodes correspond to record types. If there is a link between two record types T_1 and T_2, and the link is many-one from T_1 to T_2, then we draw an arc from the node T_1 to that for T_2 and we say the link is from T_1 to T_2. Nodes and arcs are labelled by the names of their record types and links.

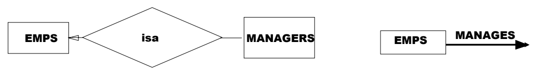


Representing Entity Sets in the Network Model

Entity sets are represented directly by logical record types; the attributes of an entity set become fields of the logical record format. The only special case is when an entity set E forms its key with fields of some entity set F, to which E is related through relationship R. We do not need to place those fields of F in the record format for E, because the records of E do not need to be distinguished by their field values. Rather, they will be distinguished by the physical pointers placed in the records of E to represent the relationship R, and these pointers will lead from a record e of type E to the corresponding record of type F that holds the key value for e.

Alternatively, when the relationship concerned is isa, and the subset has no field that the superset does not have, (as between MANAGERS and EMPS), we could eliminate the record type for the subset, e.g. MANAGERS, altogether, and let the relationships between MANAGERS and other entity sets (besides EMPS) be represented in the network model by links involving EMPS.

The isa relationship itself could be represented by a one-bit field telling whether an employee is a manager. Another choice is to represent the isa implicitly; only EMPS records that represent managers will participate in relationships, such as MANAGES, that involve the set of managers.

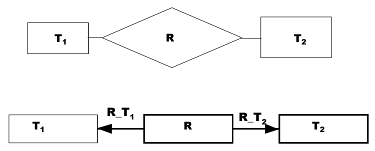


Representing Relationships

Among relationships, only those that are binary and many-one (or one-one as a special case) are representable directly by links. However, we can use the following trick to represent arbitrary relationships. Say we have a relationship R among entity sets $E_1,E_2,\ldots,E_k$. We create a new logical record type T representing k-tuples $(e_1,e_2,\ldots,e_k)$ of entities that stand in the relationship R. The format for this record type might be empty. However, there are many times when it is convenient to add information-carrying fields in the format for the new record type T. In many events, we create links $L_1,L_2,\ldots,L_k$. Link L_i is from record type T to the record type for entity set E_i, which we shall also call E_i. The intention is that the record of type T for $(e_1,e_2,\ldots,e_k)$ is linked to the record of type E_i for e_i, so each link is many-one.

As a special case, if the relationship is many-one from $E_1,E_2,\ldots,E_{k-1}$ to E_k, and furthermore, the entity set E_k does not appear in any other relations, then we can identify the record type T with E_k, storing the attributes of E_k in T.

***Example*: The relationship SUPPLIES is many-one from SUPPLIERS and ITEMS to PRICE, and PRICE participates in no relationship but this one. We may therefore create a type T with links to ITEMS and SUPPLIERS, and containing PRICE as a field.**



***Example*: A purely many-many relationship is between courses and students with the intended meaning that the student is taking the course. To represent this relationship in the network model, we would use two entity sets, COURSES and STUDENTS, each with appropriate fields, such as**

COURSES (DEPT, NUMBER, INSTRUCTOR)
STUDENTS (ID#, NAME, ADDRESS, STATUS)

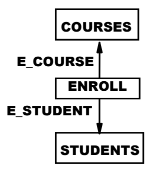
We need to introduce a new record type, say ENROLL, that represents single pairs in the relationship set, i.e., one course and one student enrolled in that course. There might not be any fields in ENROLL, or we might decide to use ENROLL records to store information that really does refer to the pair consisting of a course and a student, e.g., the grade the student receives in the course.

Thus, we might use record format

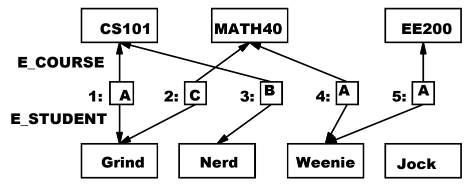
ENROLL (GRADE)

Notice that two or more enrollment records may look the same, in the sense that they have the same values in their GRADE fields. They are distinguished by their addresses, i.e., by their "object identity".
We also need two links, one from ENROLL to COURSES, which we shall call E_COURSE, and one from ENROLL to STUDENTS, which we shall cal E_STUDENT.

The link E_COURSE associates with each ENROLL record a unique COURSES record, which we take to be the course in which the enrollment is made. Likewise, E_STUDENT associates with each ENROLL record a unique STUDENTS record, that of the student who is thereby enrolled. Each student record is said to own the enrollment record which the link associates to that student.



The network



Physical connections representing links

For example, ENROLL record 1 represents only the fact that student Grind is enrolled in CS101. The record for Grind owns ENROLL records 1 and 2. Weenie owns 4 and 5, while Jock owns no enrolled records. CS101 owns ENROLL records 1 and 3. There is no conflict that Grind also owns record 1, because their ownership is through different links.

That is Grind is the owner of 1 according to the E_STUDENT link and CS101 the owner of that record according to the E_COURSE link.

***Example*: Let us design a network for the YVCB database scheme. We start with logical record types for the six entity sets that remain after excluding MANAGERS, which as we mention above, can be represented by the logical record type for its superset, EMPS. Thus, we have logical record formats:**

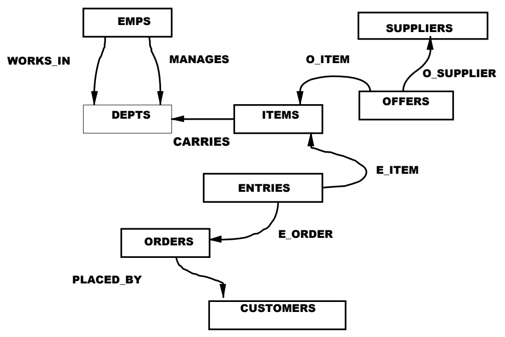
EMPS (ENAME, SALARY)
DEPTS (DNAME, DEPT#)
SUPPLIERS (SNAME, SADDR)
ITEMS (INAME, ITEM#)
ORDERS (O#, DATE)
CUSTOMERS (CNAME, CADDR, BALANCE)

We need two more record types, because two of the relationships, SUPPLIES and INCLUDES, are not binary, many-one relationships. Let us use record type ENTRIES to represent order-item-quantity facts. It makes sense to store the quantity in the entity record itself, because the relationship INCLUDES is many-one from ORDERS and ITEMS to QUANTITY. Thus, we need only links from ENTRIES to ITEMS and ORDERS, which we call E_ITEM and E_ORDER, respectively.
Similarly, a new record type OFFERS can serve to represent the facts of the SUPPLIES relation. We prefer to store PRICE as a field of OFFERS, for the same reason as was discussed above concerning QUANTITY. We shall use O_ITEM and O_SUPPLIER, as the links from OFFERS to ITEMS and SUPPLIERS, respectively.

The last two record types for our network are thus:

ENTRIES (QUANTITY)
OFFERS (PRICE)

The relationships, other than SUPPLIES and INCLUDES, are many-one and binary. Thus, they are directly representable by links.
The only special remark needed is that the relationship MANAGES, originally between DEPTS and MANAGERS, will now be between DEPTS and EMPS, since we agreed to use EMPS to represent managers. Since this relation is one-one, we could have it run in either direction, and we have chosen to have it run from EMPS to DEPTS.



Comparison of Network and Relation Schemes: Link-Following Operations on Networks

Records can have build-in, invisible pointers that represent declared links.

***Example*: One day YVCB owner Simon De Lamb gets curious and wants to know whether some customer has a balance that exactly equals the price of some items. He has only to say**

$$\pi_{CNAME}\left(CUSTOMERS \underset{BALANCE=PRICE}{\infty} SUPPLIES\right)$$

in relational algebra.

However, in the network model, whose languages only allow us to follow links, there is really no convenient way to compare customers' balances with items' prices. When we do follow links, we could relate customers to the items they have ordered by an expression like

PLACED_BY (E_ORDER (E_ITEM (ITEMS)))

The requirement for equality between the O# fields of INCLUDES and ORDERS was hidden by out use of the natural join. However, natural join can only be used where the attributes have the same name in the relation schemes; real relational database system do not support the natural join directly, requiring it to be expressed as an equijoin, with the explicit equality of values spelled out.

There is one important advantage to the relational model. The result of an operation on relations is a relation, so we can build complex expressions of relational algebra easily. However, the result of operations on network is not a network, because the pointers and unique identifiers for records cannot be referred to in network query languages. Thus, new networks cannot be constructed by queries; they must be constructed by the data definition language.

There is an additional distinction between the network and relational models in the way they treat many-many relationships. In the network model there are forbidden (they can be replaced by several many-one relationships).