The Hierarchical Data Model

A hierarchy is simply a network that is a forest (collection of trees) in which all links point in the direction from child to parent. Any entity-relational diagram can be represented in the hierarchical model.

We introduce virtual record types.

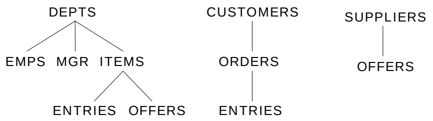
A Simple Network Conversion Algorithm

We must start at a node with as many incoming links as possible and make it the root of a tree. We attach to that tree all the nodes that can be attached, remembering that links must point to the parent. When we can pick up no more nodes this way, we start with another, unattached node as a root, and attach as many nodes to that as we can. Eventually, each node will appear in the forest one or more times, and at this point we have a hierarchy.

```
procedure BUILD (n)
        make n selected;
        for each link from some node m to n do begin
                make m a child of n;
                if m is not selected then BUILD (m)
        end
end
/* main program */
make all nodes unselected;
while not all nodes are selected do begin
        pick an unselected node n;
        /* prefer a node n with no links to unselected
           nodes and prefer a node with many incoming
           links */
        BUILD (n)
end
```

Simple hierarchy-building procedure

Example: Consider the network of our example. DEPTS is a good candidate to pick as the first node, because it has three incoming links, two from EMPS and one from ITEMS. We then consider EMPS, but find it has no incoming links. However, ITEMS has incoming links from ENTRIES and OFFERS. These have no incoming links, so we are done building the tree with root DEPTS. All the above mentioned nodes are now selected. The remaining nodes with no outgoing links are CUSTOMERS and SUPPLIERS. If we start with CUSTOMERS, we add ORDERS as a child and ENTRIES as a child of ORDERS, but can go no further. From SUPPLIERS we add OFFERS as a child and are done. Now, all nodes are selected, and we are finished building the forest. The two children of DEPTS that come from node EMPS, we have changed one, that representing the manager of the department, to MGR.

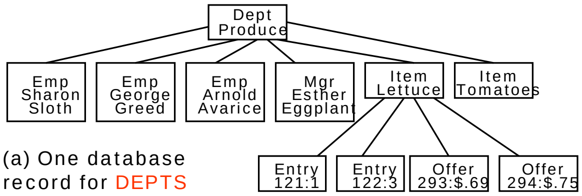


First attempt at a hierarchy for the YVCB database scheme

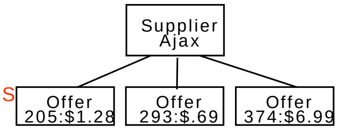
Database Records

Hierarchies of logical record types are scheme level concepts. The instances of the database corresponding to a scheme consist of a collection of trees whose nodes are records; each tree is called a database record. A database record corresponds to some one tree of the database scheme, and the root record of a database record corresponds to one entity of the root record type. If T is a node of the scheme, and S is one of its children, then each record of type T in the database record has zero or more child records of type S.

Example:



(a) One database record for DEPTS



(b) One database record for SUPPLIERS

Figure (a) shows one database record for the DEPTS tree. This database record's root corresponds to the Produce Department, and it should be understood that the entire database instance has database records similar to this one for each department. The instance also includes a database record for each customer, with a structure that is an expansion of the middle tree and it includes a database record for every supplier, with the structure implied by the rightmost tree. An example, for supplier Ajax, is shown in (b).

We see the Produce Department record at the root (a). There are three children of the Produce Department record, for the three employees of that department, Sloth, Greed, and Avarice.

Corresponding to the child MGR of DEPTS is one child of Produce, that for Ester Eggplant, the manager of the department. While we expect to find many employee children, there would normally be only one manager record, the DEPTS-MGR relationship is one-to-one. Finally, we see two children of the Produce record corresponding to items sold: lettuce and tomatoes.

Each ITEMS record has some ENTRIES children and some OFFERS children. We have shown two of each for lettuce, but none for tomatoes – a node can translate into zero records of that type in a given database record. For records representing entries and offers, we have indicated the unique identifier that distinguishes each such record from all others of the same type; e.g. ENTRIES record 121 has QUNATITY 1.

Recall that entries have only a quantity, and offers only a price as real data, and thus we cannot differentiate among records of these types by field values alone. Records for departments, employees, and so on, are uniquely identified by the values in their fields. As in networks, these unique identifiers may be thought of as the addresses of the records.

Record Duplication

Certain record types, namely ENTRIES and OFFERS, appear twice in the hierarchical scheme. An offer record by supplier s to sell item i appears both as a child of the ITEMS record for i and as a child of the SUPPLIERS record for s. OFFERS record 293 appears twice and we can deduce thereby that this offer is an offer by Ajax to sell lettuce at $.69. This duplication causes several problems:

1. We waste space because we repeat the data in the record several times.
2. There is potential inconsistency, should we change the price in one copy of the offer, but forget to change it in the other.

The solution is found in virtual record types and pointers.

Operations in the Hierarchical Model

In the hierarchical model links are presumed to go only one way, from parent to child. We can find all OFFERS children of the lettuce ITEMS record, but how could we determine what items Ajax offers to sell? The general operation is to find the root of a database record with a specified key – for example "Ajax". We can then go from the SUPPLIERS record Ajax to all its offers, examine the entire collection of DEPTS database records, until we find the OFFERS record with a given unique identifier, say 293. This solution is too time consuming and we need pointers that lead directly where we decide they are needed.

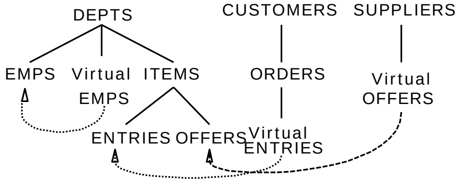
Virtual Record Types

In each scheme, we insist on having only one occurrence of any record type. Any additional places where we would like that record to appear, we place instead a virtual record of that type. In an instance, instead of a physical record, we place a pointer to the one occurrence of that physical record in the database.

```
procedure BUILD (n)
        make n selected;
        for each link from some node m to n do
                if m is not selected then begin
                        make m a child of n;
                        BUILD (m)
                end
                else /* m was previously selected */
                        make virtual m be a child of n
end
```

Example: The ENTRIES node in the tree for CUSTOMERS and the OFFERS node in the tree for SUPPLIERS will be replaced by virtual ENTRIES and virtual OFFERS, respectively, and in database tree, they will point to the corresponding record in the tree for DEPTS. Thus, in place of the record 293 in (b) would be a pointer to the record 293 in (a). This modification immediately removes the redundancy of records, and since we now have only one copy of any record to update, it removes the inconsistency, as well.

***Example*:**

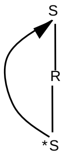


Second attempt at a hierarchy

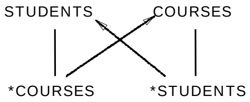
The nodes labeled ENTRIES and OFFERS in DEPTS tree remain as they were, because those nodes represent the first times these record types are encountered by the BUILD procedure.
We have replaced the MGR node by virtual EMPS. We use a pointer to an employee record, because we need a reference to a particular employee record to mark which employee is the manager of the department.

Representation of Bidirectional Relationships

Virtual record types also solve the problem of traversing links in both directions. If we have a many-one relationship from record type R to record type S, we can make R be a child of S, and then make virtual S be a child of R. If we have a many-many relationship between R and S, we cannot make either a child of the other, but we can let R and S each take their natural position in the forest, and then create a child of each that is a virtual record version of the other.



Example: Reconsider example, which discussed a many-many relationship between courses and students.



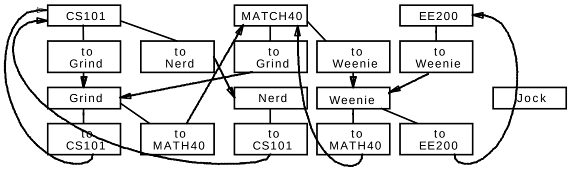
STUDENTS

COURSES

*COURSES

*STUDENTS

Representing a many-many relationship



CS101

MATCH40

EE200

to Grind

to Nerd

to Grind

to Weenie

to Weenie

Grind

Nerd

Weenie

Jock

to CS101

to MATH40

to CS101

to MATH40

to EE200

Physical connection representing virtual records

We can find the students enrolled in a given course (such as CS101) as follows:

1.Find the courses record for CS101. Finding a root record, given its key, is one of the typical operations of a hierarchical system.

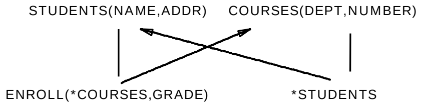
2.Find all the virtual STUDENTS children of CS101. We would find pointers to the STUDENTS records for Grind and Nerd, but at this point, we would not know the names or anything about the students to whose records we have pointers.

3.Follow the pointers to find the actual student records and the names of these students.

Combined Record Types

To navigate quickly along arbitrary paths, we need combined records consisting of some data fields, and pointer fields that point to other record types.

Example: Suppose we want to store grades in the enrollment records that interpose between student and course records. We replace the Virtual COURSES child of STUDENTS by a combined record that has a Virtual COURSES field as well as a GRADE field. *T stands for a virtual record type of T.



STUDENTS(NAME,ADDR)

COURSES(DEPT,NUMBER)

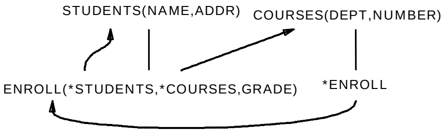
ENROLL(*COURSES,GRADE)

*STUDENTS

Scheme with combined record type

To find all the grades issued in CS101, we have to find the root of the COURSES database record for CS101, then follow all the virtual student pointers and from them, find their enrollment children.

There are several other schemes. If we don't want to duplicate enrollments as children of both STUDENTS and COURSE. We can go directly from the CS101 record to its enrollments, and find the grades directly. To find all the students taking CS101 we need to go first to ENROLL, then to STUDENTS, via two virtual record pointers.



STUDENTS(NAME,ADDR)

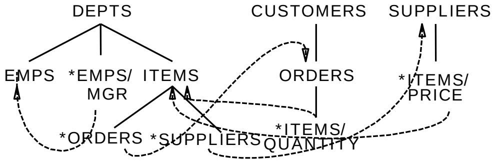
COURSES(DEPT,NUMBER)

ENROLL(*STUDENTS,*COURSES,GRADE)

*ENROLL

Another scheme for courses and students

Example: Entries, with their quantities, and offers, with their prices, and handled by the trick of the previous example, using combined records. We have also added virtual ORDERS as a child of ITEMS, to facilitate finding the orders for a given item, and we have similarly added virtual SUPPLIERS as a child of ITEMS to help find out who supplies a given item.



DEPTS

CUSTOMERS

SUPPLIERS

EMPS

*EMPS/ MGR

ITEMS

ORDERS

*ITEMS/ PRICE

*ORDERS

*SUPPLIERS

*ITEMS/ QUANTITY

Improved design for YVCB database

We don't add virtual DEPTS as a child of either EMPS or MGR, because the only way to reach EMPS or MGR records is through their DEPT, and therefore, we shall "know" the department anyway, without needing to follow a pointer.