Chapter 2
Database and Expert System Technology

2.1 Hierarchical Model

The hierarchical data model is a logical schema and can be viewed as a subset of a network model because it imposes a further restriction on the relationship types in the form of an inverted tree structure. The linkage between record types is in an automatic fixed set membership. The database access path of a hierarchical database follows the hierarchical path from a parent to child record. The default path is a hierarchical sequence of top-to-bottom, left-to-right, and front-to-back.

It is common that many real life data can be structured in hierarchical form. For example, enrollment in a university can be ordered according to the department organizations. As hierarchies are familiar in nature and in human society, it seems natural to represent data in a hierarchical structure. Data represent ideas about the real world that people conceive in terms of entities. Based on the characteristics of entities, entity type can be defined. Figure 2.1 shows a generic hierarchical tree that represents entity types where entities refer to record types and record. In the tree, the record type at the top is usually known as the “root.” Record types are groups of entities or records that can be described by the same set of attributes. In general, the root may have any number of dependents; each of these may have any number of lower-level dependents, and so on, to any number of levels. Individual records are the actual occurrences of data. The right-hand side is the hierarchical sequence.

There are some important properties of the hierarchical database model.

There is a set of record types (R₁, R₂, ..., Rₙ). It is possible to designate a field of record type as an identifier of a record occurrence of this type. This may provide either a unique or a nonunique identification. This identifier is called a key.

• There is a set of relationships connecting all record types in one data structure diagram.
• There is not more than one relationship between any two record types Rᵢ and Rⱼ. Hence, relationships need not be labeled.
• The relationships expressed in the data structure diagram form a tree with all edges pointing towards the leaves.
• Each relationship is 1:n and it is total. That is, if Rₐ is the parent of Rᵦ in the hierarchy, then for every record occurrence of Rᵦ there is exactly one Rₐ record connected to it.

To construct a hierarchical model, it is natural to build an entity–relationship (ER) model and map it to a hierarchical model because an ER model carries more semantics. Once an ER model is built, if relationships are all binary, we can map a 1:n or 1:1 relationship from A to B as a binary tree. To map a m:n relationship from A to B, we can use virtual record types (pointer to actual records) which are distinguished by an identifier (ID) field in a physical address as shown in Fig. 2.2 (McElreath 1981).

2.1.1 Hierarchical Data Definition Language

Two types of structures are used to implement the inverted tree structure of a hierarchical model, namely data definition and data occurrence trees. The role of a data definition tree is to describe the data types and their relationships. For example Fig. 2.1 shows seven data types, in a parent (the top one)–child (the bottom one) relationship with respect to each other. Data occurrence tree represents the actual data in the database. Figure 2.1 shows fifteen data occurrences in hierarchical sequence, the default read sequence in hierarchical model.
Due to the limitation of an inverted tree structure, the hierarchical model cannot be used to implement the following:

- m:n relationship between two record types
- A child record type with more than one parent record type
- n-ary relationships with more than two participating record types

With virtual pointers architecture, each record type can be in an m:n relationship with another record type through the pointers. The record type with the source pointers is called the logical child. Its target record type is called the logical parent. For example, Fig. 2.2 shows that the record types of Cpointers and Lpointers are logical child record types. Their corresponding logical parent record types are Customers and Loan Contracts. As a result of these pointers, Record type Customers and Loan Contracts are in an m:n relationship such that each customer can sign many loan contracts, and each loan contract can be signed by many customers.

### 2.1.2 Hierarchical Data Manipulation Language (HDML)

HDML is a record-at-a-time language for manipulating hierarchical databases. The commands of a HDML must be embedded in a general-purpose programming language, called a host language. Following each HDML command, the last record accessed by the command is called the current database record. The system maintains a pointer to the current record. Subsequent database commands proceed from the current record, and move to a new current record depending on the operation of the command. The traversal of the database access follows the inverted tree structure, i.e., each database navigation path according to the hierarchical sequence. For example, Fig. 2.1 has five access paths as follows:
The commands in any Data Manipulation Language (DML), can be divided into two sets, retrieval and modification commands. The following are the syntax of the hierarchical DML of IMS (IBM’s Information Management System, a hierarchical DBMS). There are four parameters in IMS DML. They are:

- Function Code, which defines the database access function
- Program Control Block, which defines the external subschema access path
- I-O-Area, which is a target segment address
- Segment Search Argument, which defines the target segment selection criteria as follows:

CALL “CBLTDLI” USING FUNCTION-CODE
PCB-MASK
I-O-AREA
SSA-1
...
SSA-n.

Note: CBLTDLI is a call by a COBOL program to access the DL/1 database.

Retrieval Command:

- Get Unique (GU)
  This command retrieves the leftmost segment that satisfies the specified condition. For example, the following Get unique command is to retrieve a Loan Balance segment of a loan with the loan contract number 277988 and loan balance date of July 22, 1996.

CALL “CBLTDLI” USING GU
PCB-MASK
I-O-AREA
LOAN_CONTRACT# = 277988
BALANCE_DATE = ‘960722’
Hierarchical Model

• Get Next (GN)
  This command retrieves the next segment based on the preorder traversal algorithm from the current location. The clause for the record identifier and retrieval conditions is optional. If the clause is not given, GET NEXT would retrieve the next sequential segment from the current location. For example, the following command is to retrieve the next Loan Contract record after the current Loan Contract record occurrence.

  \[
  \text{CALL "CBLTDLI" USING GN} \\
  \text{PCB-MASK} \\
  \text{LOAN_CONTRACT.}
  \]

• Get Next WITHIN PARENT (GNP)
  This command retrieves segments from the set of children sharing the same parent as the current segment of the given type. The parent segment is visited by a previous GET command, i.e., it establishes parentage of a segment type according to the current pointer of its parent segment type. For example, the following command retrieves the next in a hierarchical sequence of a Loan Interest segment under the Loan Contract segment type with a loan_contract# of “277988”.

  \[
  \text{CALL "CBLTDLI" USING GNP} \\
  \text{PCB-MASK} \\
  \text{LOAN_INTEREST} \\
  \text{LOAN_CONTRACT# = 277988.}
  \]

Hierarchical Modification Commands:

• INSERT (ISRT)
  This command stores a new segment and connects it to a parent segment. The parent segment must be selected by the previous GET command. For example, the following commands are to insert a balance segment of $1,000,000 under the Loan_contract number 277988 on July 22, 1996.

  \[
  \text{CALL "CBLTDLI" USING GN} \\
  \text{PCB-MASK} \\
  \text{LOAN_CONTRACT}.
  \]
CALL “CBLTDLI” USING GU
PCB-MASK
I-O-AREA
LOAN_CONTRACT# = 277988.

MOVE “19960722” TO BALANCE_DATE.
MOVE 1000000 TO BALANCE_AMOUNT.
CALL “CBLTDLI” USING ISRT
PCB-MASK
LOAN_BALANCE.

• REPLACE (REPL)
This command replaces the current segment with the new segment. It can be used to alter the detail of the current segment. For example, the following commands are to update the loan balance of loan_contract# 277988 from 1,000,000 to 2,000,000 on July 22, 1996. The GHU function is a get hold unique call to apply a record lock on a segment before an update.

CALL “CBLTDLI” USING GHU
PCB-MASK
I-O-AREA
LOAN_CONTRACT# = 277988
BALANCE_DATE = ‘960722’

MOVE 2000000 TO BALANCE_AMOUNT.
CALL “CBLTDLI” USING REPL.

• DELETE (DELT)
This command physically deletes the current segment and all of its child segments. For example, the following command deletes a balance segment of a loan_contract# 277988 on July 22, 1996.

CALL “CBLTDLI” USING GHU
PCB-MASK
I-O-AREA
LOAN_CONTRACT# = 277988
BALANCE_DATE = ‘960722’

CALL “CBLTDLI” USING DELT.
The Network model is a logical schema and is based on tables and graphs (CODASYL 1971). The nodes of a graph (segment types) usually correspond to the entity types, which are represented as connections (sets) between tables in the form of network. The insertion and retention of segment types depend on the set membership constraints that exist between the owner and member segments, with automatic or manual insertion, and fixed, mandatory, or optional retention.

A network database model is similar to a hierarchical database model that represents data and a data relationship in a graphical form. The network model differs from the hierarchical model as:

- There can be more than one edge between a given pair of entities
- There is no concept of root and
- A segment can have more than one parent segment

For example, Fig. 2.3 is a network model for the university enrollment system.
The CODASYL (Network) model is composed of two basic data constructs: the record and the set, respectively. These two data constructs are built up from simpler data elements which are discussed as follows:

- **Data Item**—An occurrence of the smallest unit of named data. It is represented in the database by a value. A data item may be used to build other more complicated data constructs. This corresponds to an attribute in the ER data model.
- **Data Aggregation**—An occurrence of a named collection of data items within a record.
- **Record**—An occurrence of a named collection of data items or data aggregates. This collection is in conformity with the record type definition specified in the database schema.
- **Set**—An occurrence of a named collection of records. A set occurrence is in direct correspondence with the set type definition specified in the database schema. Each set type consists of one owner record type and at least one member record type.
- **Area**—The notion of an area used to identify the partition of record occurrences. An area is a named collection of records that need not preserve owner–member relationships. An area may contain occurrences of one or more record types and a record type may have occurrences in more than one area.

While designing a network database, the following rules must be followed to ensure the integrity of the definitions:

- An area is a named subdivision of the database.
- An arbitrary number of areas may be defined in a system.
- Records may be independently assigned to areas of their set associations.
- A record occurrence is stored within one area only.
- A single occurrence of a set type may span several areas.
- Each set type must be uniquely named and must have an owner record type. A special type of set which has exactly one occurrence and for which the system is the owner may be declared as a singular set.
- Any record type may be defined as the owner of one or more set types.
- If a set has an owner record which has no member record, the set is known as empty or null.
- A record cannot be used as an owner record in more than one occurrence of the same set type.
- A record cannot be participated as a member record in more than one occurrence of the same type.
- A set contains exactly one occurrence of its owner.
- A set may have any number of member occurrences.

The following shows some record entries and set entries of the university enrollment system.
RECORD NAME IS DEPARTMENT WITHIN ANY AREA
    KEY DEPARTMENTID IS DEPARTMENT#
    DUPLICATES ARE NOT ALLOWED
    CALL CHECK-AUTHORIZATION BEFORE DELETE
    DEPARTMENT# TYPE IS NUMERIC INTEGER
    DEPARTMENT-NAME TYPE IS CHARACTER 30

RECORD NAME IS INSTRUCTOR WITHIN ANY AREA
    KEY INSTRUCTORID IS INSTRUCTOR-NAME
    DUPLICATES ARE ALLOWED
    CALL CHECK-AUTHORIZATION BEFORE DELETE
    INSTRUCTOR-NAME TYPE IS CHARACTER 30
    INSTRUCTOR-ADDRESS TYPE IS CHARACTER 40

SET NAME IS HIRE
    OWNER IS DEPARTMENT
    ORDER IS PERMANENT INSERTION IS FIRST
    MEMBER IS INSTRUCTOR
    INSERTION IS AUTOMATIC RETENTION IS MANDATORY
    SET SELECTION IS THRU HIRE OWNER IS IDENTIFIED
        BY APPLICATION

The INSERTION clause specifies the class of membership of a member record in a set type. There are two options in this clause, AUTOMATIC and MANUAL. For the AUTOMATIC option, the system ensures the status of the member record in the occurrences of the set type. For the MANUAL option, the application must handle the record as a member of some set occurrence in the database. The RETENTION is concerned with the ways in which records retain their membership in the database. There are three ways; FIXED, MANDATORY, and OPTIONAL, for handling set membership. For the FIXED option, if a record occurrence is made a member in a set, then that record must exist as a member of the set in which it associates. For MANDATORY, if a record is made a member in some set, then it must exist as a member of some occurrence of this set type. Therefore, it is possible to transfer the record from one set occurrence to another. For OPTIONAL, a record is allowed to be moved from a set occurrence without requiring that the record be placed in a different occurrence.

2.2.1 Network Data Definition Language

As shown in Fig. 2.4, the database task group (DBTG) specification proposes three levels of data organization. There are two pairs of Data Definition Language (DDL)
and DML for the schema level and subschema level respectively. The four languages are:

- The schema DDL
- The subschema DDL
- The DML and
- The Data Storage Description Language (DSDL)

The schema is the logical description of the global database and is made up of a description of all the areas, set types, and record types as well as associated data items and data aggregates. A database is defined as consisting of all areas, records, and sets that are controlled by a specific schema. A schema definition consists of the following elements:

- A schema entry
- One or more area entries
- One or more record entities and
- One or more set entries

The schema must be mapped to the physical storage device. This transformation is achieved by declaring the physical properties of the schema in the DSDL. The use of the DDL and DSDL provide the database management system (DBMS) with a certain degree of data independence. In the DDL, the schema and area entries are

---

**Fig. 2.4** Architecture of a CODASYL database task group (DBTG) system
Information Systems Reengineering, Integration and Normalization
Fong, J.
2015, XVII, 391 p. 211 illus., Softcover
ISBN: 978-3-319-12294-6