Chapter A: Network Model

- Basic Concepts
- Data-Structure Diagrams
- The DBTG CODASYL Model
- DBTG Data-Retrieval Facility
- DBTG Update Facility
- DBTG Set-Processing Facility
- Mapping of Networks to Files

Basic Concepts

- Data are represented by collections of records.
  - similar to an entity in the E-R model
  - Records and their fields are represented as record type

Type customer = record
type account = record
customer-name: string; account-number: integer;
customer-street: string; balance: integer;
customer-city: string;
end
end

- Relationships among data are represented by links
  - similar to a restricted (binary) form of an E-R relationship
  - Restrictions on links depend on whether the relationship is many-
  - many, many-to-one, or one-to-one.

Data-Structure Diagrams

- Schema representing the design of a network database.

- A data-structure diagram consists of two basic components:
  - Boxes, which correspond to record types.
  - Lines, which correspond to links.

- Specifies the overall logical structure of the database.

Data-Structure Diagrams (Cont.)

- For every E-R diagram, there is a corresponding data-structure diagram.

- Since a link cannot contain any data value, represent an E-R
  - relationship with attributes with a new record type and links.

General Relationships

- To represent an E-R relationship of degree 3 or higher, connect
  - the participating record types through a new record type that is
  - linked directly to each of the original record types.

1. Replace entity sets account, customer, and branch with record
   types account, customer, and branch, respectively.

2. Create a new record type Rlink (referred to as a dummy record
   type).

3. Create the following many-to-one links:
   - CustRlink from Rlink record type to customer record type
   - AcctRlnk from Rlink record type to account record type
   - BrncRlnk from Rlink record type to branch record type

Network Representation of Ternary Relationship

- All links are treated as many-to-one relationships.
- To model many-to-many relationships, a record type is defined to
  - represent the relationship and two links are used.
**DBTG Sets**

- The structure consisting of two record types that are linked together is referred to in the DBTG model as a **DBTG set**.
- In each DBTG set, one record type is designated as the **owner**, and the other is designated as the **member** of the set.
- Each DBTG set can have any number of set occurrences (actual instances of linked records).
- Since many-to-many links are disallowed, each set occurrence has precisely one owner, and has zero or more member records.
- No member record of a set can participate in more than one occurrence of the set at any point.
- A member record can participate simultaneously in several set occurrences of different DBTG sets.

**Repeating Groups**

- Provide a mechanism for a field to have a set of values rather than a single value.
- Alternative representation of weak entities from the E-R model.
- Example: Two sets.
  - customer (customer-name)
  - customer-address (customer-street, customer-city)
- The following diagrams represent these sets without the repeating-group construct.

**DBTG Data-Retrieval Facility**

- The DBTG data manipulation language consists of a number of commands that are embedded in a host language.
- **Run unit** — system application program consisting of a sequence of host language and DBTG command statements. Statements access and manipulate database items as well as locally declared variables.
- **Program work-area** (or **user work area**) — a buffer storage area the system maintains for each application program.

**DBTG Variables**

- **Record Templates**
- **Currency pointers**
  - Current of record type
  - Current of set type
  - Current of run unit
- **Status flags**
  - **DB-status** is most frequently used
  - Additional variables: DB-set-name, DB-record-name, and DB-data-name

**Example Schema**

- Templates for three record types: customer, account, and branch.
- Six currency pointers
  - Three pointers for record types: one each to the most recently accessed customer, account, and branch record
  - Two pointers for set types: one to the most recently accessed record in an occurrence of the set depositor; one to the most recently accessed record in an occurrence of the set account
  - One run-unit pointer.
- Status flags: four variables defined previously
- Following diagram shows an example program work area state.
The Find and Get Commands

- **find** locates a record in the database and sets the appropriate currency pointers.
- **get** copies of the record to which the current of run-unit points from the database to the appropriate program work area template.

Example: Executing a **find** command to locate the customer record belonging to Johnson causes the following changes to occur in the state of the program work area.

- The current of the record type customer now points to the record of Johnson.
- The current of set type depositor now points to the set owned by Johnson.
- The current of run unit now points to customer record Johnson.

Access of Individual Records

- **find any** <record type> using <record-field>
  Locates a record of type <record type> whose <record-field> value is the same as the value of <record-field> in the <record type> template in the program work area.

- Once such a record is found, the following currency pointers are set to point to that record:
  - The current of run-unit pointer
  - The record-type currency pointer for <record type>
  - For each set in which that record belongs, the appropriate set currency pointer.

- **find duplicate** <record type> using <record-field>
  Locates (according to a system-dependent ordering) the next record that matches the <record-field>.

Access of Records Within a Set

- Other **find** commands locate records in the DBTG set that is pointed to by the <set-type> currency pointer.
- **find first** <record type> within <set-type>
  Locates the first database record of type <record type> belonging to the current <set-type>.
- To locate the other members of a set, k we use
  - **find next** <record type> within <set-type>
    which finds the next element in the set <set-type>.
- **find owner within** <set-type>
  Locates the owner of a particular DBTG set.

Predicates

- For queries in which a field value must be matched with a specified range of values, rather than to only one, we need to:
  - **get** the appropriate records into memory
  - examine each one separately for a match
  - determine whether each is the target of our **find** statement

Example DBTG Query

- Print the total number of accounts in the Perryridge branch with a balance greater than $10,000.

```sql
count := 0;
branch.branch-name := "Perryridge";
find any branch using branch-name;
find first account within account-branch;
while DB-status = 0 do
  begin
    get account
    if account.balance > 10000 then count := count + 1;
    find next account within account-branch;
  end;
print (count);
```

DBTG Update Facility

- DBTG mechanisms are available to update information in the database.
- To create a new record of type <record type>:
  - insert the appropriate values in the corresponding <record type> template
  - add this new record to the database by executing
    ```sql
    store <record type>
    ```
- Can create and add new records only one at a time.

DBTG Update Facility (Cont.)

- To modify an existing record of type <record type>:
  - find that record in the database
  - get that record into memory
  - change the desired fields in the template of <record type>
  - reflect the changes to the record to which the currency point of <record type> points by executing
    ```sql
    modify <record type>
    ```

- To delete an existing record of type <record type>:
  - make the currency pointer of that type point to the record in the database to be deleted
  - delete that record by executing
    ```sql
    erase <record type>
    ```
- Delete an entire set occurrence by finding the owner of the set and executing
  ```sql
  erase all <record type>
  ```
  - Deletes the owner of the set, as well as all the set’s members.
  - If a member of the set is an owner of another set, the members of that second set also will be deleted.
  - ```sql
  erase all <record type>
  ```
**DBTG Set-Processing Facility**

- Mechanisms are provided for inserting records into and removing records from a particular set occurrence.
- Insert a new record into a set by executing the `connect` statement:
  \[
  \text{connect} \hspace{1em} \text{<record type>} \hspace{1em} \text{to} \hspace{1em} \text{<set-type>}
  \]
- Remove a record from a set by executing the `disconnect` statement:
  \[
  \text{disconnect} \hspace{1em} \text{<record type> \hspace{1em} from} \hspace{1em} \text{<set-type>}
  \]

**Example disconnect Query**

- Close account A-201, that is, delete the relationship between account A-201 and its customer, but archive the record of account A-201.
- The following program removes account A-201 from the set occurrence of type depositor. The account will still be accessible in the database for record-keeping purposes.
  
  ```sql
  account.account-number := "A-201";
  find for update any account using account-number.
  get account, find owner within depositor. disconnect account from depositor.
  ```

**DBTG Set-Processing Facility (Cont.)**

- To move a record of type `<record type>` from one set occurrence to another set occurrence of type `<set-type>`:
  1. Find the appropriate record and the owner of the set occurrences to which that record is to be moved.
  2. Move the record by executing
    \[
    \text{reconnect} \hspace{1em} \text{<record type> \hspace{1em} to} \hspace{1em} \text{<set-type>}
    \]
- Example: Move all accounts of Hayes that are currently at the Perryridge branch to the Downtown branch.

**Example reconnect Query**

- `customer customer-name := "Hayes";`
- `find any customer using customer-name;`
- `find first account within depositor;` while DB-status = 0 do begin
  - `find owner within account-branch;`
  - `get branch;`
  - `if branch.branch-name = "Perryridge" then begin` `branch.branch-name := "Downtown";`
  - `find any branch using branch-name; reconnect account to account-branch;` end
  - `find next account within depositor;` end

**Set Insertion Example**

- Create account A535 for customer Hayes at the Downtown branch.
- Set insertion is manual for set type depositor and is automatic for set type account-branch.
- `branch.branch-name := "Downtown";`
- `find any branch using branch-name; account.account-number := "A-535"; account.balance := 0;` `store account; customer.customer-name := "Hayes";`
- `find any customer using customer-name;` `connect account to depositor;`

**DBTG Set-Processing Facility (Cont.)**

- A newly created member record of type `<record type>` of a set type `<set-type>` can be added to a set occurrence either explicitly (manually) or implicitly (automatically).
- Specify the insert mode at set-definition time via
  \[
  \text{insertion is} \hspace{1em} \text{<insert mode>}
  \]
  - manual: `connect <record type> to <set-type>`
  - automatic: `store <record type>`

**DBTG Set-Processing Facility (Cont.)**

- Restrictions on how and when a member record can be removed from a set occurrence are specified at set-definition time via
  \[
  \text{retention is} \hspace{1em} \text{<retention-mode>}
  \]
  - `<retention-mode>` can take one of the three forms:
    1. fixed — a member record cannot be removed. To reconnect a record to another set, we must erase that record, recreate it, and then insert it into the new set occurrence.
    2. mandatory — a member record of a particular set occurrence can be reconnected to another set occurrence of only type `<set-type>`.
    3. optional — no restrictions on how and when a member record can be removed from a set occurrence.
Set Ordering

Set ordering is specified by a programmer when the set is defined:

- **order** is <order-mode>.
  - **first**: A new record is inserted in the first position; the set is in reverse chronological ordering.
  - **last**: A new record is inserted in the final position; the set is in chronological ordering.
  - **next**: Suppose that the currency pointer or <set-type> points to record X:
    - If X is a member type, a new record is inserted in the next position following X.
    - If X is an owner type, a new record is inserted in the first position.

Set Ordering Example

- Add a new account A-125. For each <order-mode> option, the new set ordering is as follows:
  - **first**: {A-125,A-305,A-402,A-408}
  - **last**: {A-305,A-402,A-408,A-125}
  - **next**: Suppose that the currency pointer points to record “Turner”; then the new set order is {A-125,A-305,A-402,A-408}

- **prior**: Suppose that the currency pointer points to record A-402; then the new set order is {A-305,A-125,A-402,A-408}
- **system default**: Any arbitrary order is acceptable; thus, {A-305,A-402,A-125,A-408} is a valid set ordering
- **sorted**: The set must be ordered in ascending order with account number being the key; thus, the ordering must be {A-125,A-305,A-402,A-408}

Mapping of Networks to Files

- We implement links by adding pointer fields to records that are associated via a link.
- Each record must have one pointer field for each link with which it is associated.
- Example data-structure diagram and corresponding database.

Diagram showing the sample instance with pointer fields to represent the links. Each link is replaced by two pointers.

```
  |Hayes  |Main  |Harrison|   |  |
  |       |      |        |A-152|400|
  |Johnson|Alma  |Palo Alto|   |  |
  |       |      |         |A-181|500|
  |Smith  |North |Rye     |   |  |
  |       |      |         |A-215|700|
```

Mapping of Networks to Files (Cont.)

- Since the depositor link is many to many, each record can be associated with an arbitrary number of records (e.g., the account record would have a pointer to the customer record for each customer who has that account).
- Direct implementation of many-to-many relationships requires the use of variable length records.
- The DBTG model restricts links to be either one to one or one to many; the number of pointers needed is reduced, and it is possible to retain fixed-length records.

Mapping of Networks to Files (Cont.)

- Assume that the depositor link is one to many and is represented by the DBTG set depositor and this corresponding sample database.

```
  |Hayes  |Main  |Harrison|   |  |
  |       |      |        |A-182|400|
  |Johnson|Alma  |Palo Alto|   |  |
  |       |      |         |A-181|500|
  |Smith  |North |Rye     |   |  |
  |       |      |         |A-215|700|
```

Mapping of Networks to Files (Cont.)

- Because an account record can be associated with only one customer record, we need only one pointer in the account record to represent the depositor relationship.
- A customer record can be associated with many account records.
- Rather than using multiple pointers in the customer record, we can use a ring structure to represent the entire occurrence of the DBTG set depositor.
- In a ring structure, the records of both the owner and member types for a set occurrence are organized into a circular list.
- There is one circular list for each set occurrence (that is, for each record of the owner type).
Example Ring Structure

Modified Ring Structures

- Execute find owner via a ring structure in which every member-type record contains a second pointer which points to the owner record.

Physical Placement of Records

- To specify the storage strategy for DBTG set, add a placement clause to the definition of the member record type.
- The clause placement clustered via depositor
  will store members of each set occurrence close to one another physically on disk, if possible, in the same block.
- Store owner and member records close to one another physically on disk by adding the clause near owner.
  placement clustered via depositor near owner

Physical Placement of Records (Cont.)

- Storing member records in the same block as the owner reduces the number of block accesses required to read an entire set occurrence.

Sample Database

Two Data-Structure Diagrams

Sample Database Corresponding to Diagram of Figure A.3b

Sample Database Corresponding to Diagram of Figure A.6b