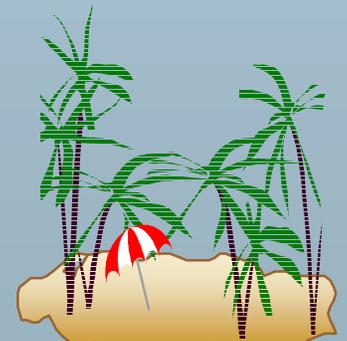# Chapter A: Network Model

- 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.
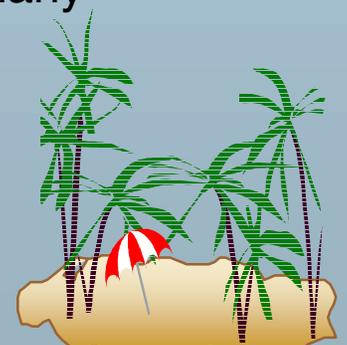
Data-Structure Diagrams

■ Schema representing the design of a network database.

■ A data-structure diagram consists of two basic components:

★ **Boxes**, which correspond to record types.
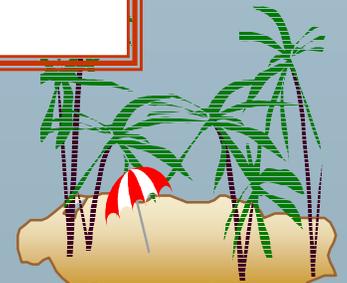
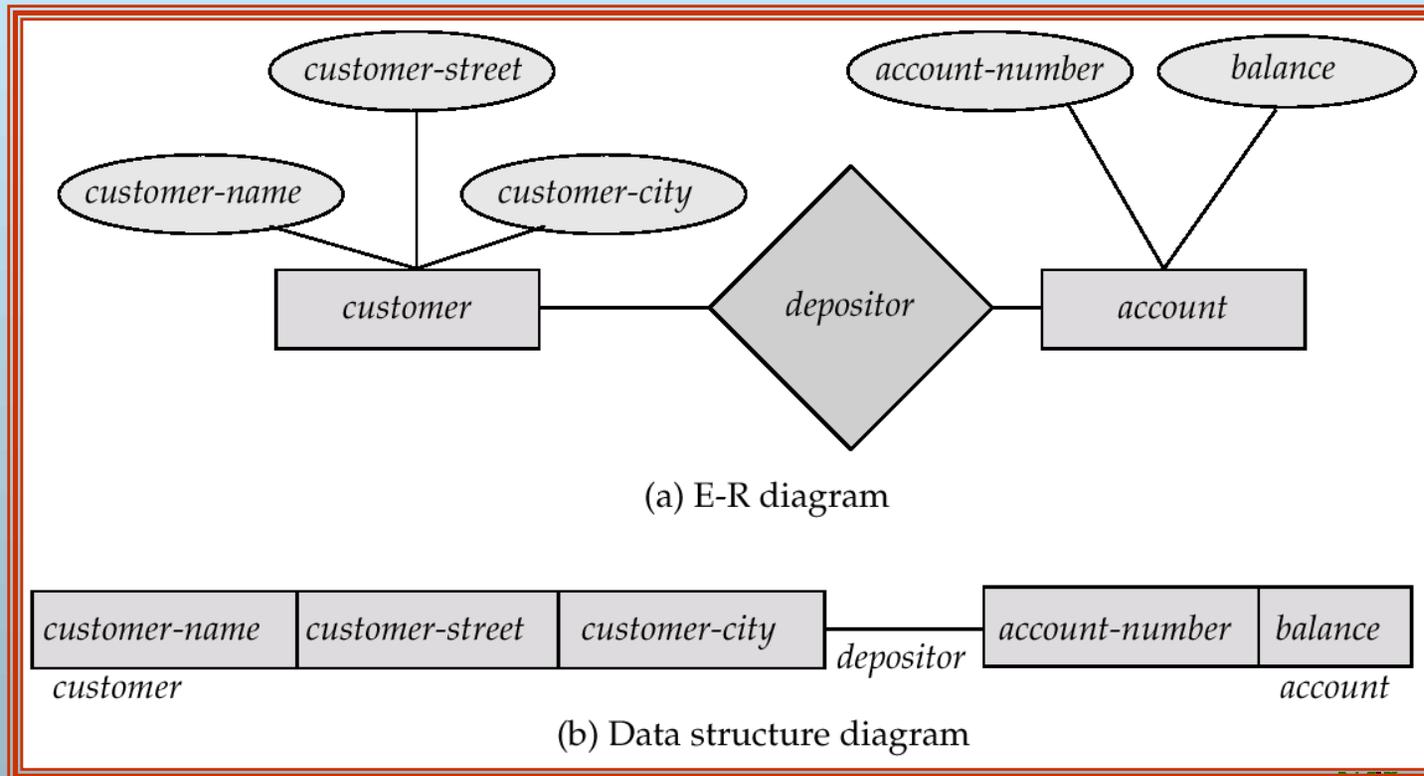★ **Lines**, which correspond to links.

■ Specifies the overall logical structure of the database.
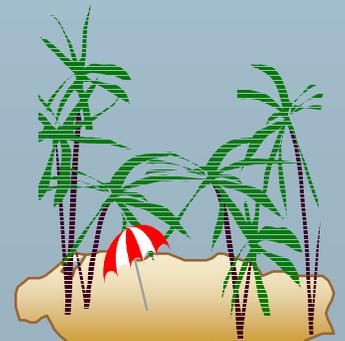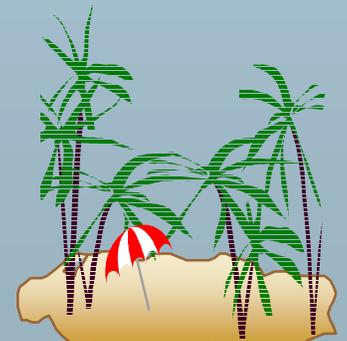
Data-Structure Diagrams (Cont.)

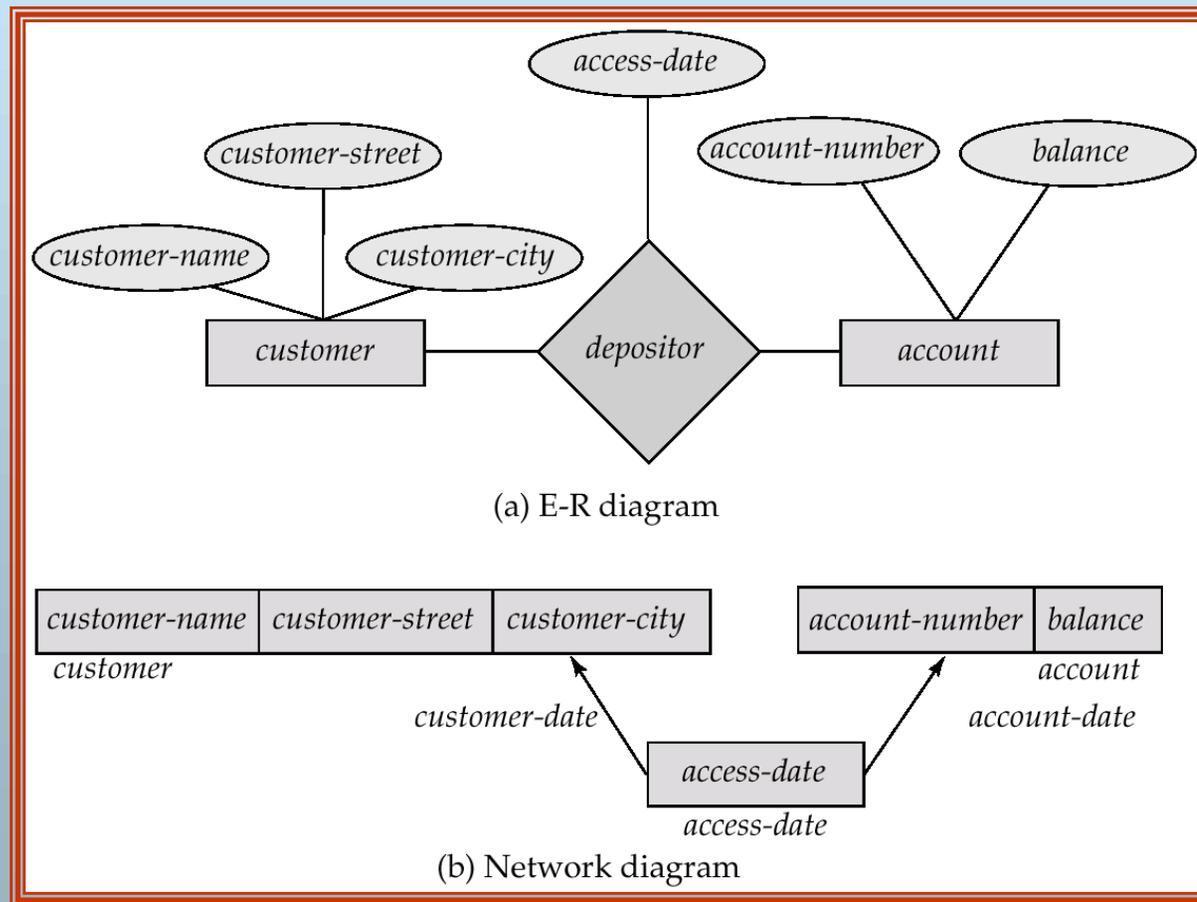- For every E-R diagram, there is a corresponding data-structure diagram.

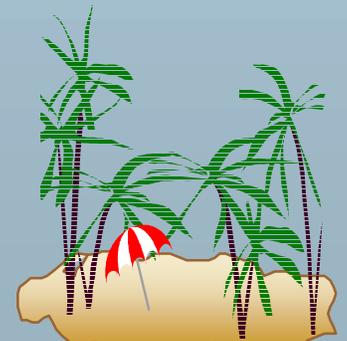(a) E-R diagram

(b) Data structure diagram

Data-Structure Diagrams (Cont.)

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

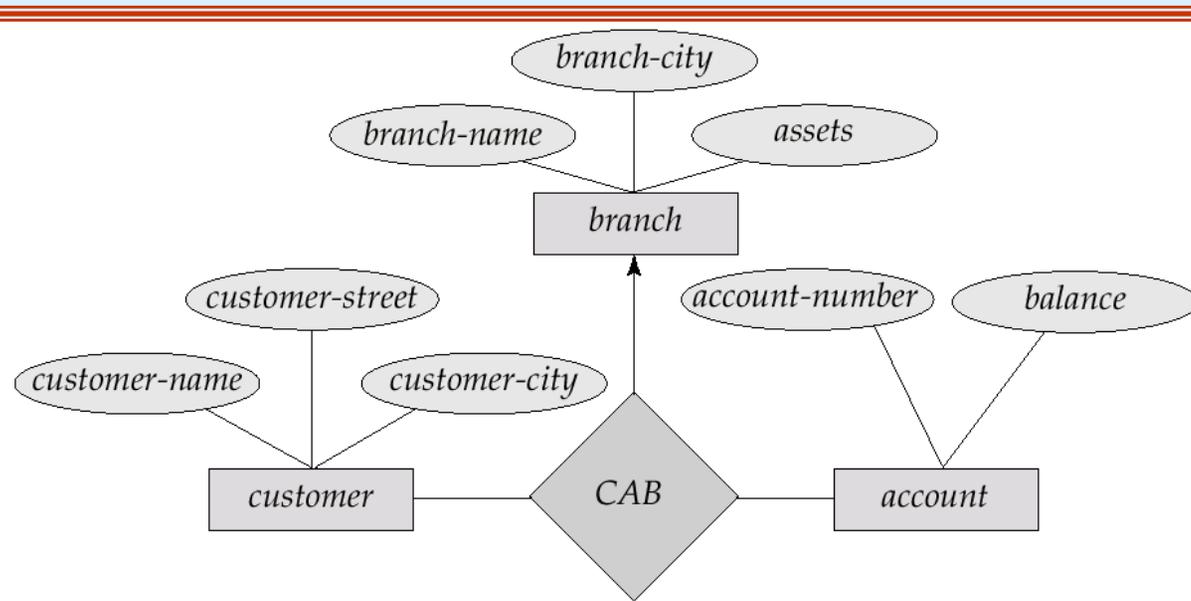
(a) E-R diagram

(b) Network diagram

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
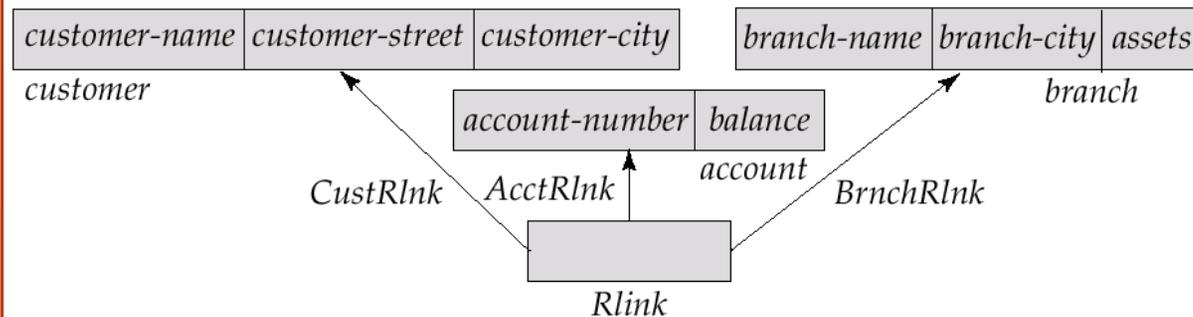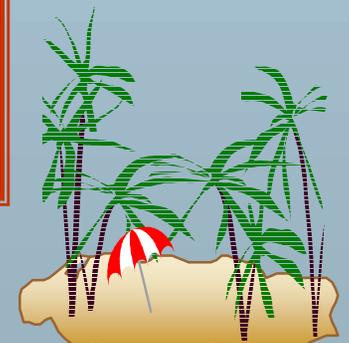
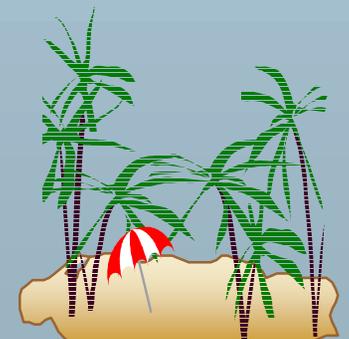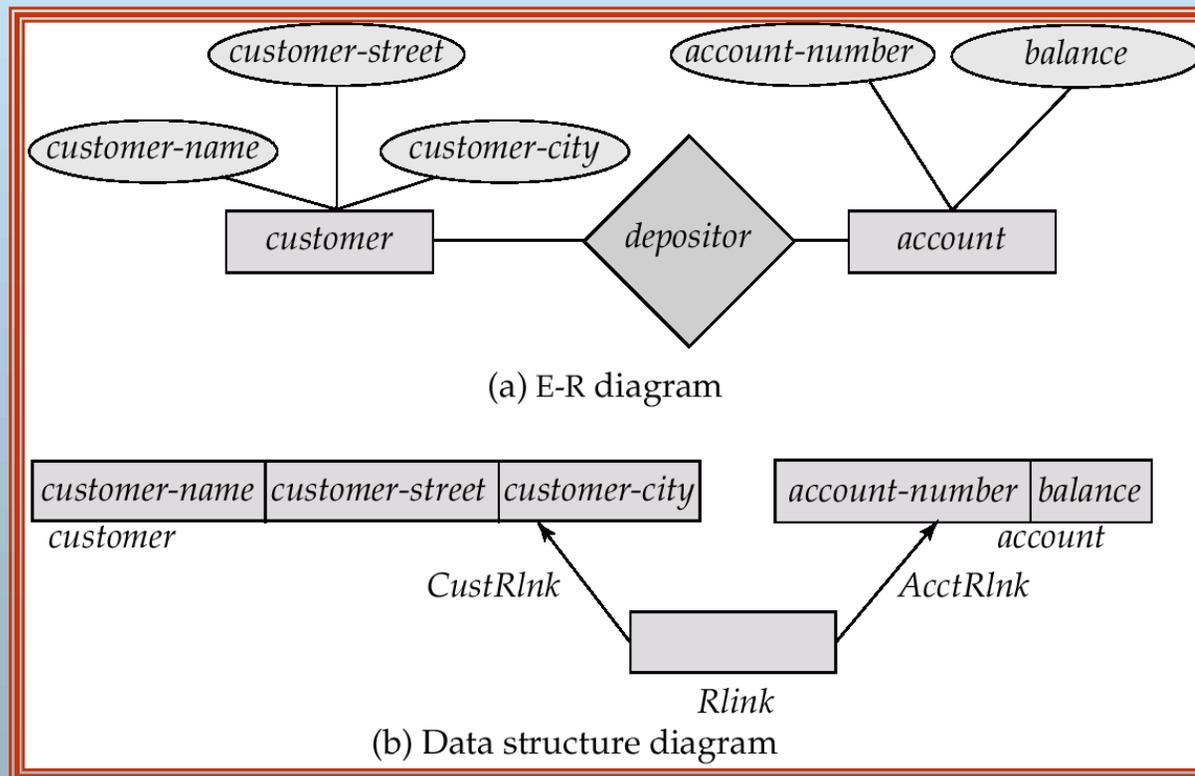# Network Representation of Ternary Relationship

(a) E-R diagram

(b) Data structure diagram

The DBTG CODASYL Model

- 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.
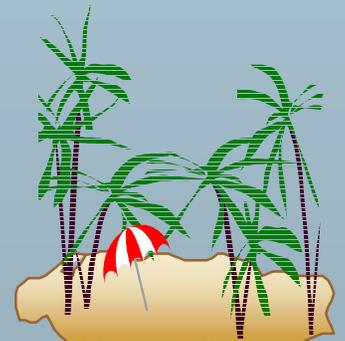
Repeating Groups

- 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.

Repeating Groups (Cont.)

(a) E-R diagram
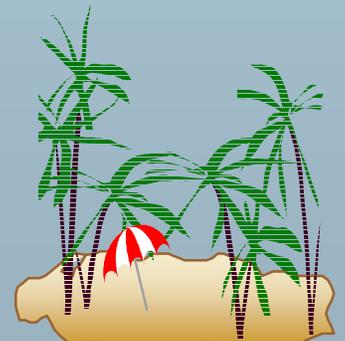
(b) Data-structure diagram
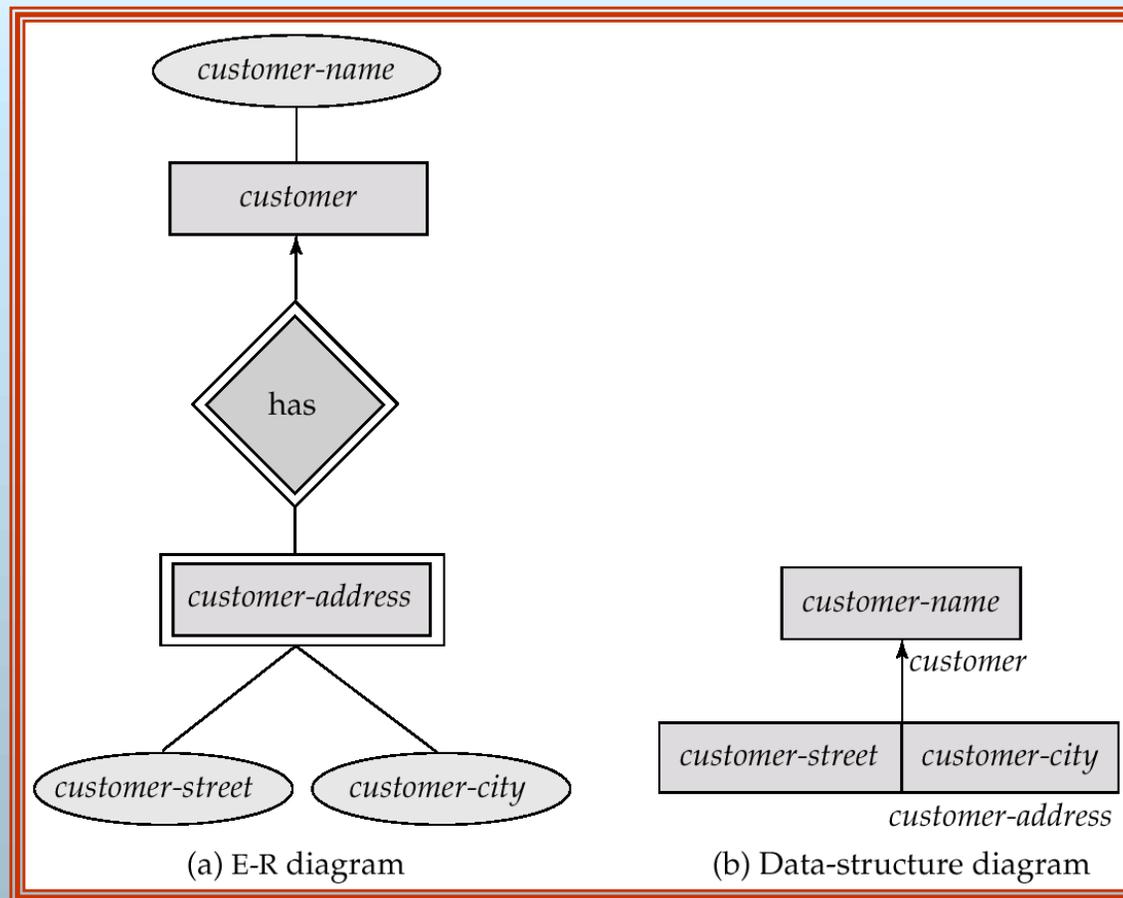
- With the repeating-group construct, the data-structure diagram consists of the single record type *customer.*

DBTG Data-Retrieval Facility

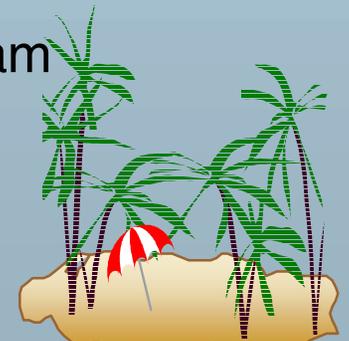■ The DBTG data manipulation language consists of a number of commands that are embedded in a host language.

■ *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
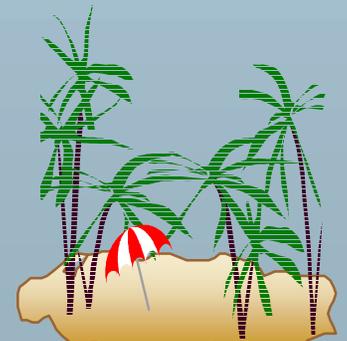
DBTG Variables

- 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

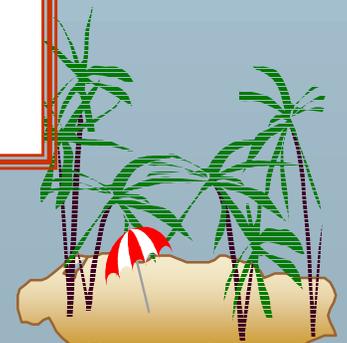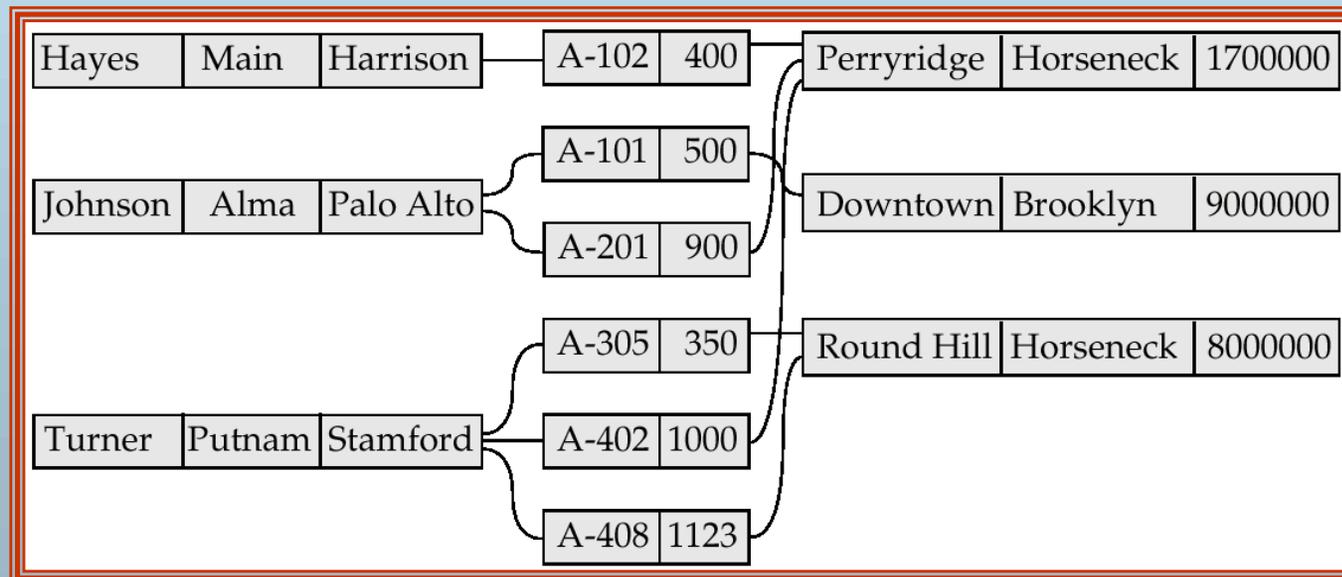| customer-name	customer-street	customer-city

customer

branch-name	branch-city	assets

branch

depositor

account-branch

account-number	balance

account

Hayes	Main	Harrison	A-102	400	Perryridge	Horseneck	1700000

A-101	500

Johnson	Alma	Palo Alto	Downtown	Brooklyn	9000000

A-201	900

A-305	350	Round Hill	Horseneck	8000000

Turner	Putnam	Stamford	A-402	1000

A-408	1123
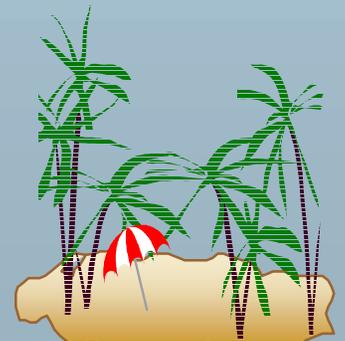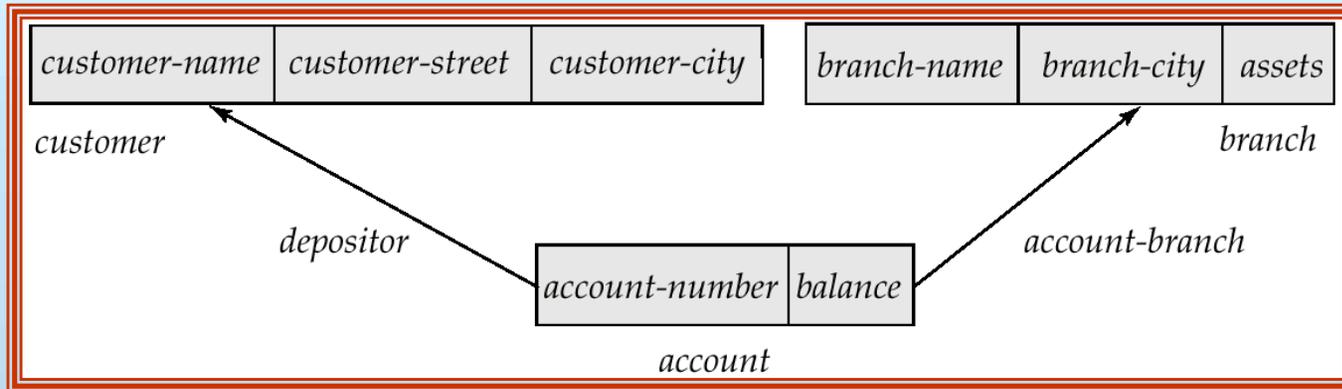
Example Program Work Area

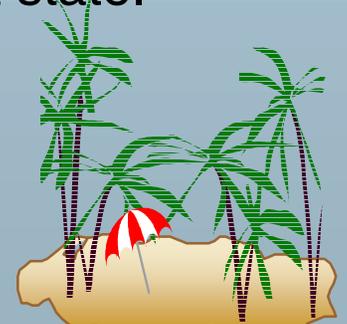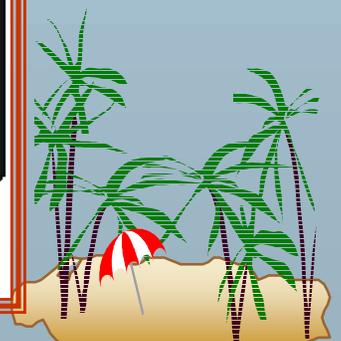■ Templates for three record types: *customer, account,* and *branch.*

■ Six currency pointers

 ★ Three pointers for record types: one each tot he 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-branch*

 ★ One run-unit pointer.

■ Status flags: four variables defined previously

■ Following diagram shows an example program work area state.

Turner	Putnam	Stamford

customer template

A-305	350

account template

Round Hill	Horseneck	8000000

branch template

Hayes	Main	Harrison

Johnson	Alma	Palo Alto

Turner	Putnam	Stamford

Perryridge	Horseneck	1700000

Downtown	Brooklyn	9000000

Round Hill	Horseneck	8000000

currency pointers

customer

account

branch

depositor

account-branch

run unit

| A-305	350		A-102	400
A-402	1000		A-101	500
A-408	1123		A-201	900

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.
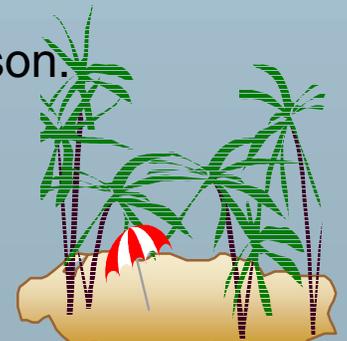
Access of Individual Records

- **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>
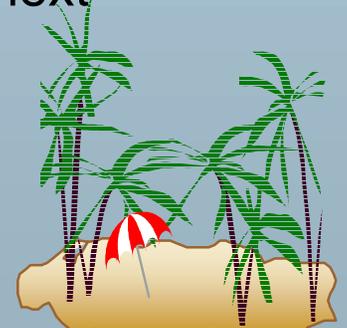
Access of Records Within a Set

- 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
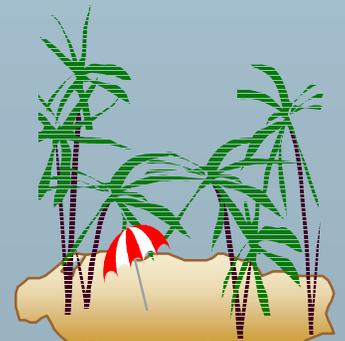
Predicates

■ 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
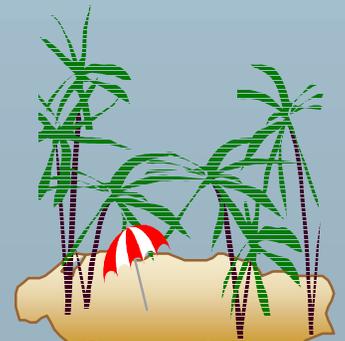
Example DBTG Query

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

  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;*
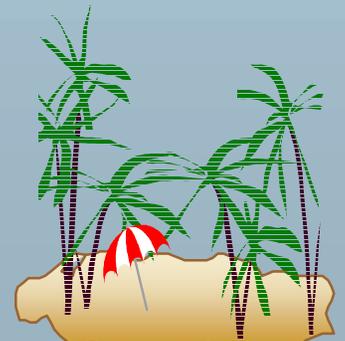   **end**
  print *(count);*
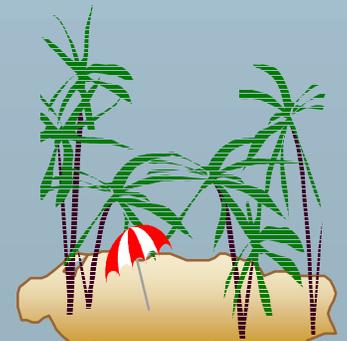
DBTG Update Facility

- 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

    store <record type>
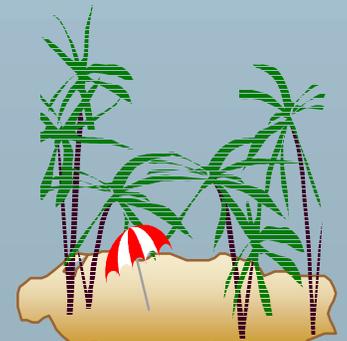
- Can create and add new records only one at a time

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

**modify** <record type>

DBTG Update Facility (Cont.)

- 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

    erase <record type>

- Delete an entire set occurrence by finding the owner of the set and executing

    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.

  ★ **erase all** is recursive.
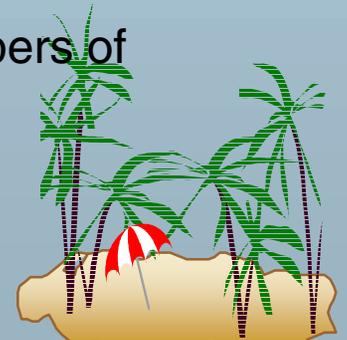
DBTG Set-Processing Facility

- 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.

     connect <record type> **to** <set-type>

- Remove a record from a set by executing the **disconnect** statement.

     disconnect <record type> **from** <set-type>
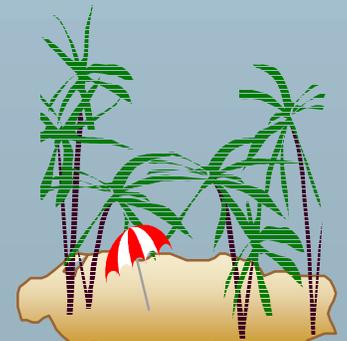
Example disconnect Query

- 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.

  account.account-number := "A-201";
  find for update any *account* **using** *account-number.*
  get *account,*
  find *owner* **within** *depositor,*
  disconnect *account* **from** *depositor.*
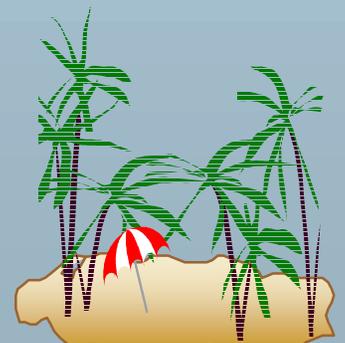
DBTG Set-Processing Facility (Cont.)

- To move a record of type <record type> from one set occurrence to another set occurrence of type <set-type>

  - ★ Find the appropriate record and the owner of the set occurrences to which that record is to be moved.

 - ★ Move the record by executing

    reconnect <record type> **to** <set-type>

- Example: Move all accounts of Hayes that are currently at the Perryridge branch to the Downtown branch.
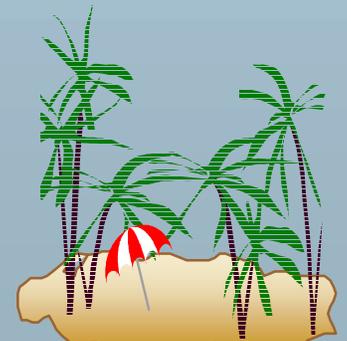
Example reconnect Query

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
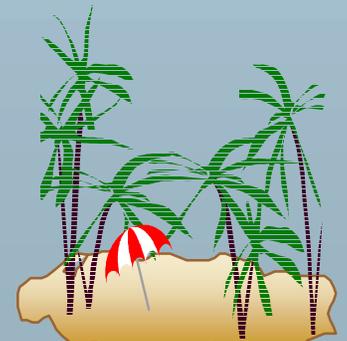
DBTG Set-Processing Facility (Cont.)

■ 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

insertion is <insert mode>

★ **manual:**   **connect** <record type> **to** <set-type>

★ **automatic:**   **store** <record type>
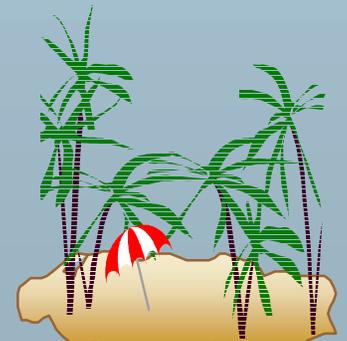
Set Insertion Example

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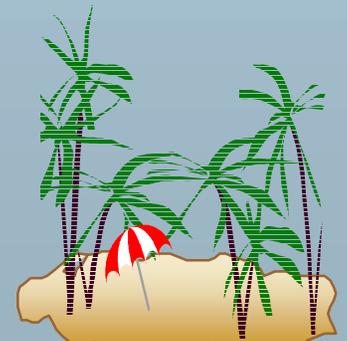
DBTG Set-Processing Facility (Cont.)

- Restrictions on how and when a member record can be removed from a set occurrence are specified at set-definition time via

  retention is <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.
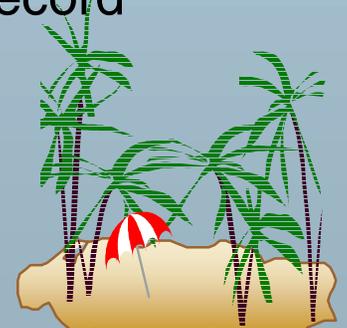
DBTG Set-Processing Facility (Cont.)

- The best way to delete a record that is the owner of set occurrence of type <set-type> depends on the specification of the set retention of <set-type>.

- **optional** — the record will be deleted and every member of the set that it owns will be disconnected. These records, however, will be in the database.

- **fixed** — the record and all its owned members will be deleted; a member record cannot be removed from the set occurrence without being deleted.

- **mandatory** — the record cannot be erased, because the mandatory status indicates that a member record must belong to a set occurrence. The record cannot be disconnected from that set.
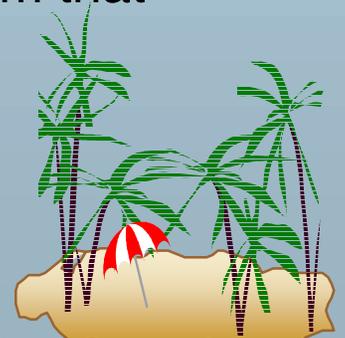
Set Ordering

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.
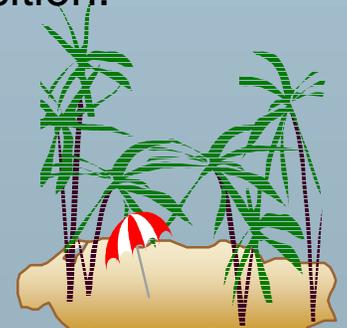
Set Ordering (Cont.)

- **prior.** If *X* is a member type, a new record is inserted in the position just prior to *X*. If *X* is an owner type, a new record is inserted in the last position.

- **system default.** A new record is inserted in an arbitrary position determined by the system.

- **sorted.** A new record is inserted in a position that ensures that the set will remain sorted. The sorting order is specified by a particular key value when a programmer defines the set.

- Example: Consider the set occurrence of type *depositor* with the owner-record customer Turner and member-record accounts A-305, A-402, and A-408 ordered as indicated in our example schema (page A.14).
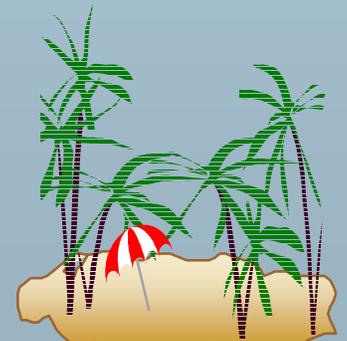
Set Ordering Example

- 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}
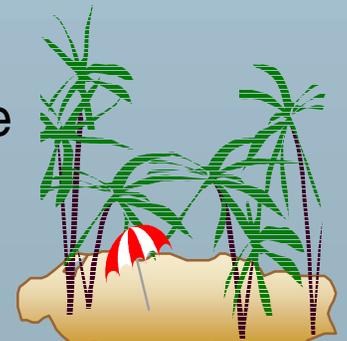
Mapping of Networks to Files

- 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.

Figure missing
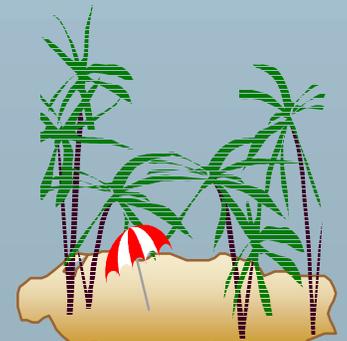
Mapping of Networks to Files (Cont.)

- Diagram showing the sample instance with pointer fields to represent the links. Each link is replaced by two pointers.
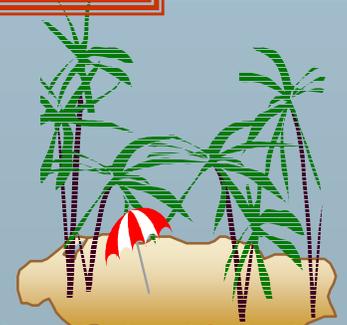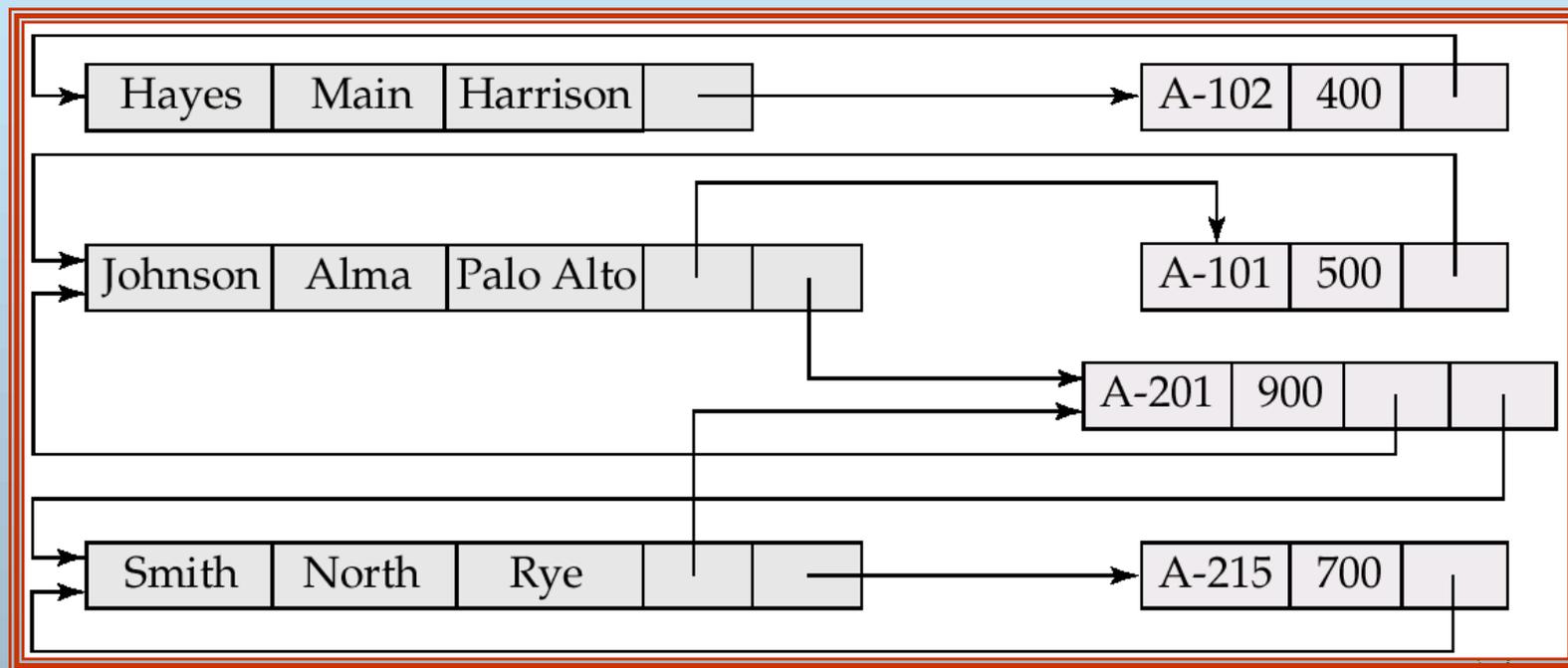
Mapping of Networks to Files (Cont.)

- 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.
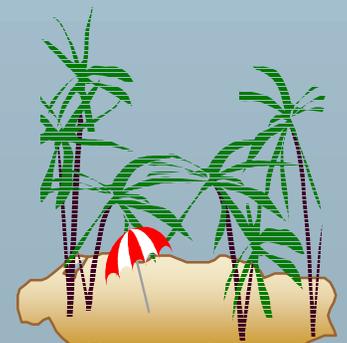
Mapping of Networks to Files (Cont.)

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

set name is *depositor*
owner is *customer*
member is *account*

Hayes	Main	Harrison		A-102	400
				A-101	500
Johnson	Alma	Palo Alto		A-201	900
Smith	North	Rye		A-215	700
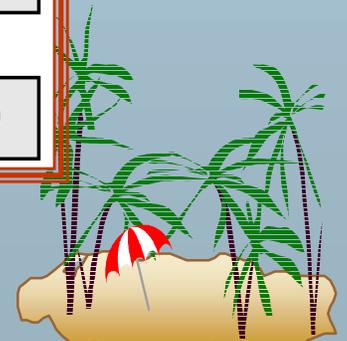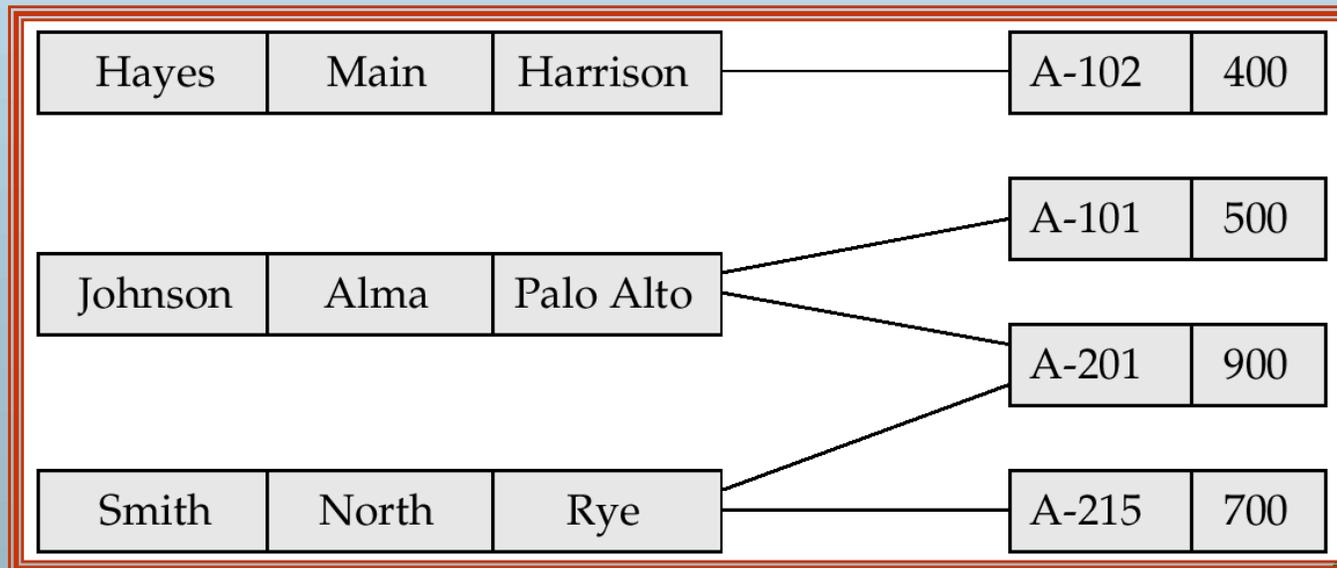
Mapping of Networks to Files (Cont.)

- 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 ant 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 an 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).
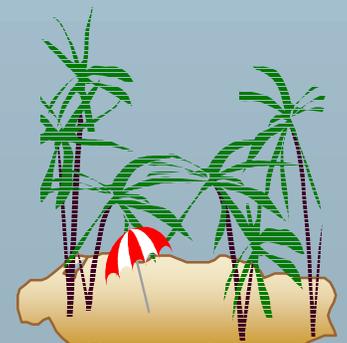
Example Ring Structure
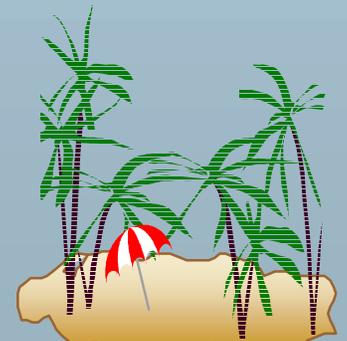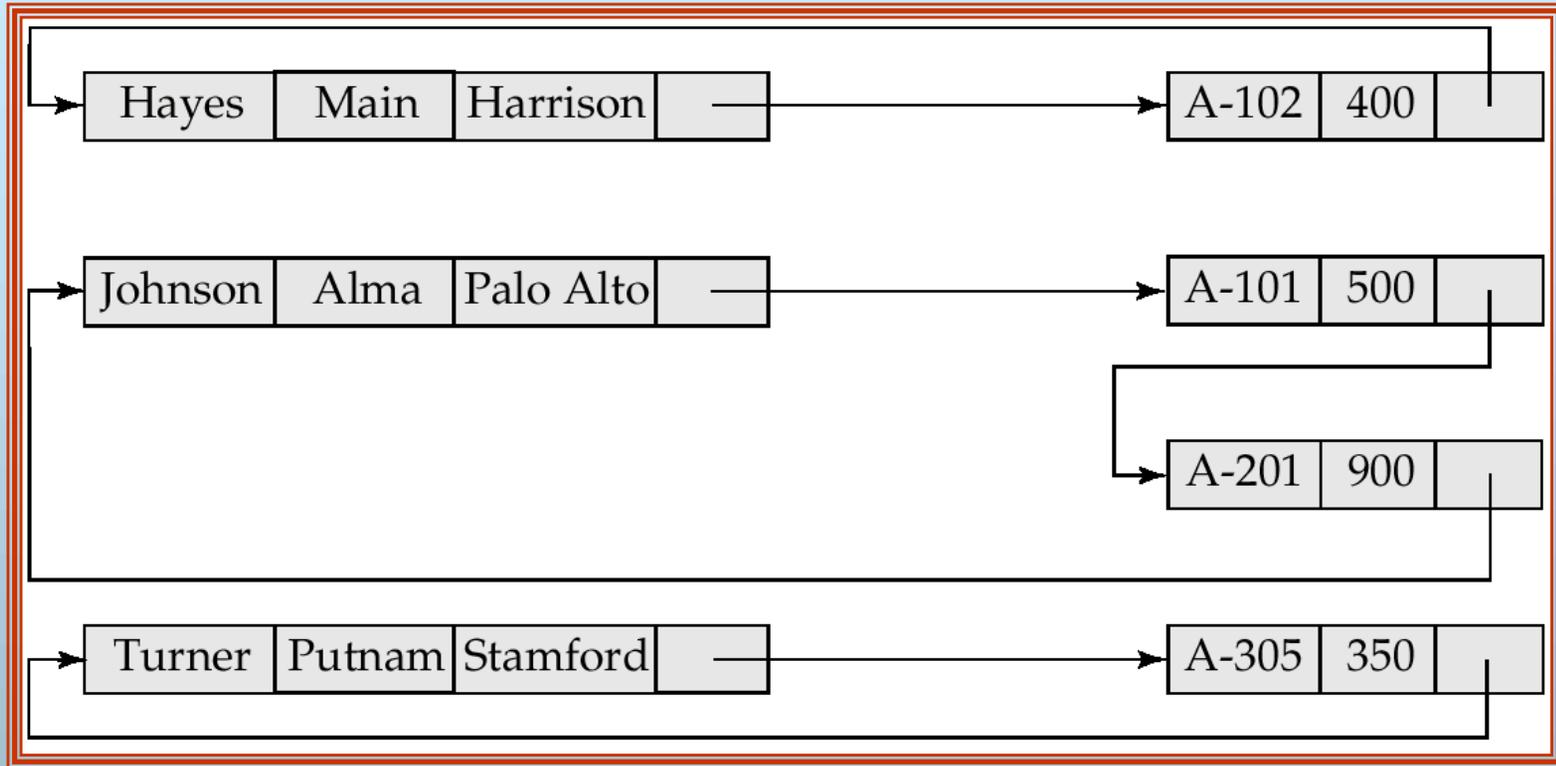
Modified Ring Structures

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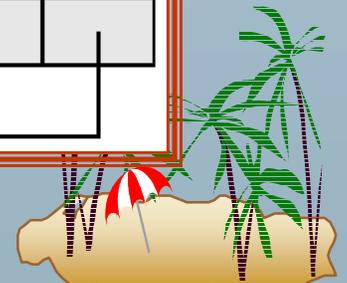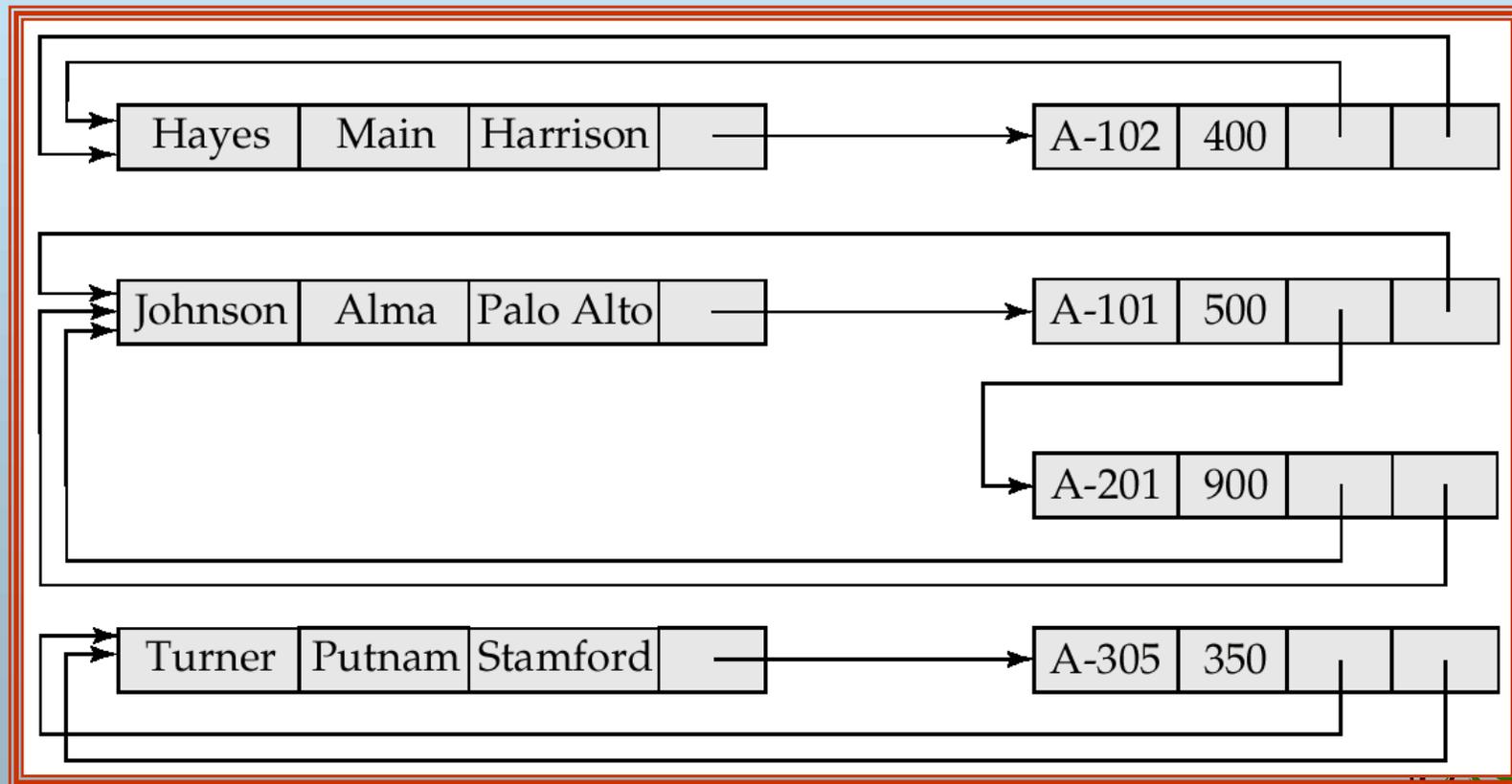
Physical Placement of Records

- 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**

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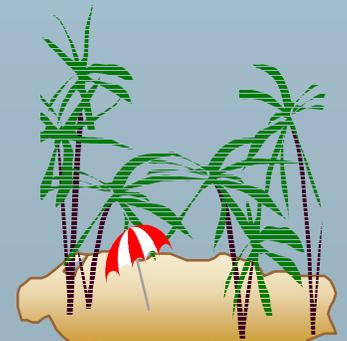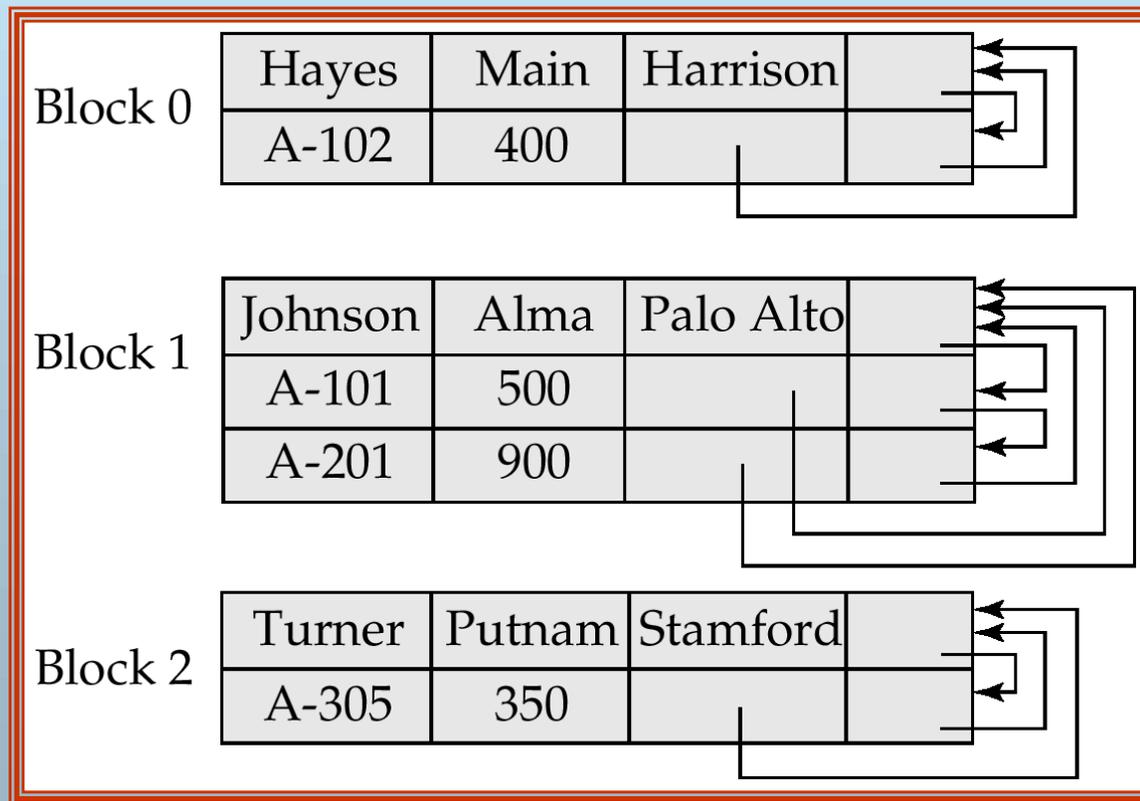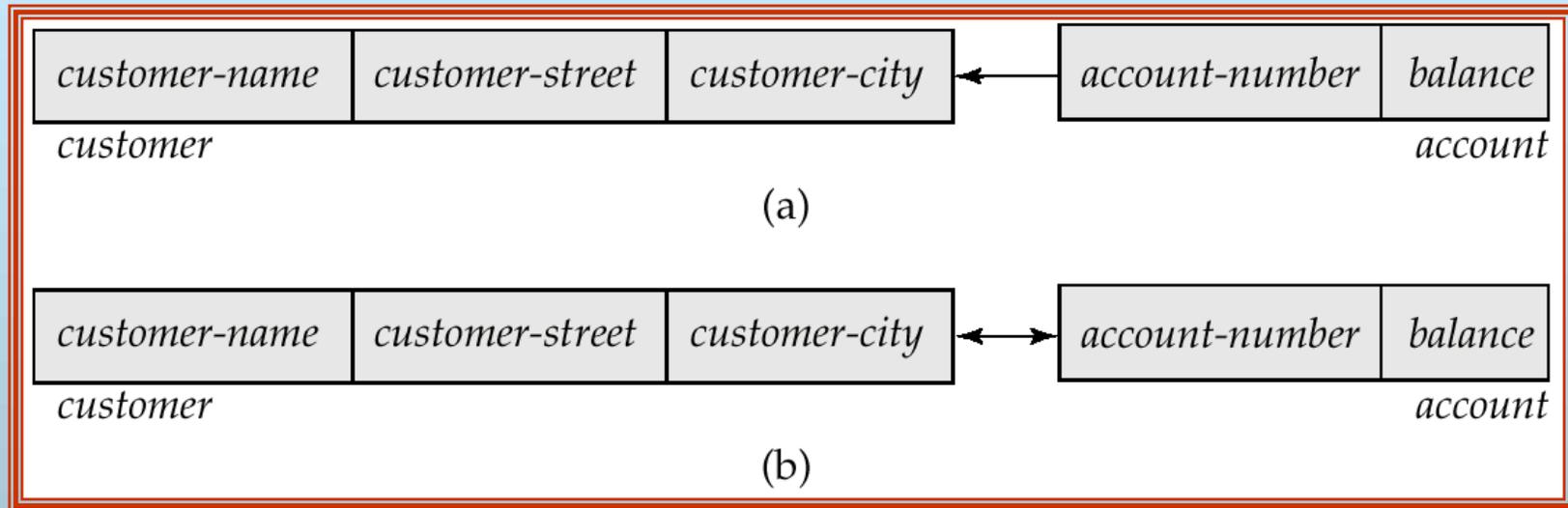

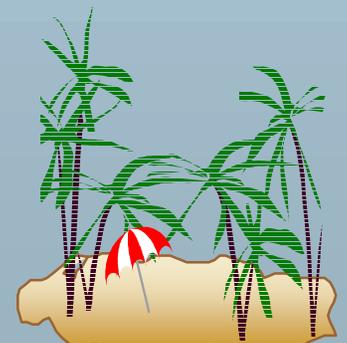|			
Block 0	Hayes	Main	Harrison
	A-102	400	

Block 1	Johnson	Alma	Palo Alto
	A-101	500	
	A-201	900	

Block 2	Turner	Putnam	Stamford
	A-305	350	

Sample Database

Hayes	Main	Harrison		A-102	400
				A-101	500
Johnson	Alma	Palo Alto			
				A-201	900
Turner	Putnam	Stamford		A-305	350

Two Data-Structure Diagrams

customer-name	customer-street	customer-city	←	account-number	balance
customer					account

(a)

customer-name	customer-street	customer-city	↔	account-number	balance
customer					account

(b)
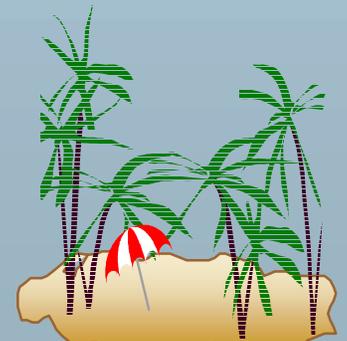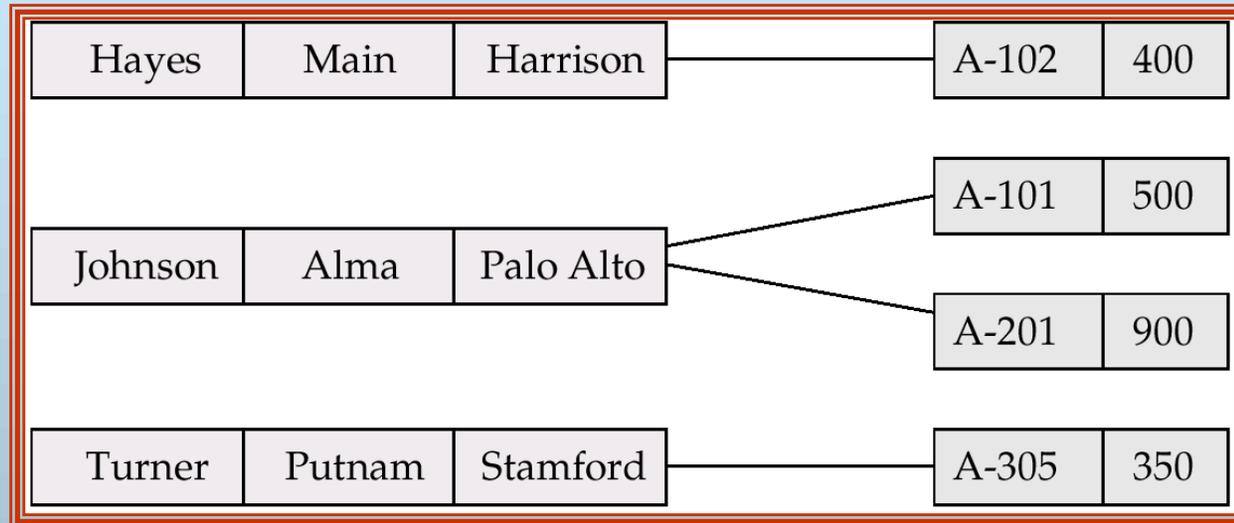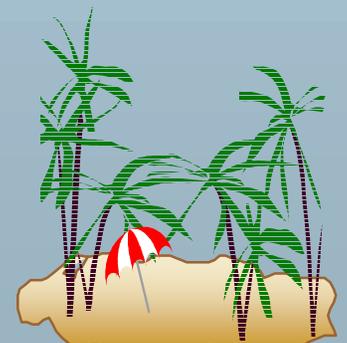
Sample Database Corresponding to Diagram of Figure A.3b

Hayes	Main	Harrison	—	A-102	400
Lindsay	Park	Pittsfield	—	A-222	700
Turner	Putnam	Stamford	—	A-305	350

Sample Database Corresponding to Diagram of Figure A.6b

Hayes	Main	Harrison		10 June 1996		A-102	400
				24 May 1996			
Johnson	Alma	Palo Alto					
				17 June 1996		A-201	900
Turner	Putnam	Stamford		28 May 1996		A-305	350
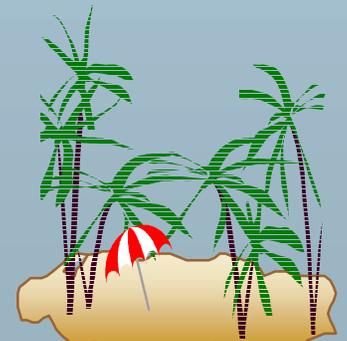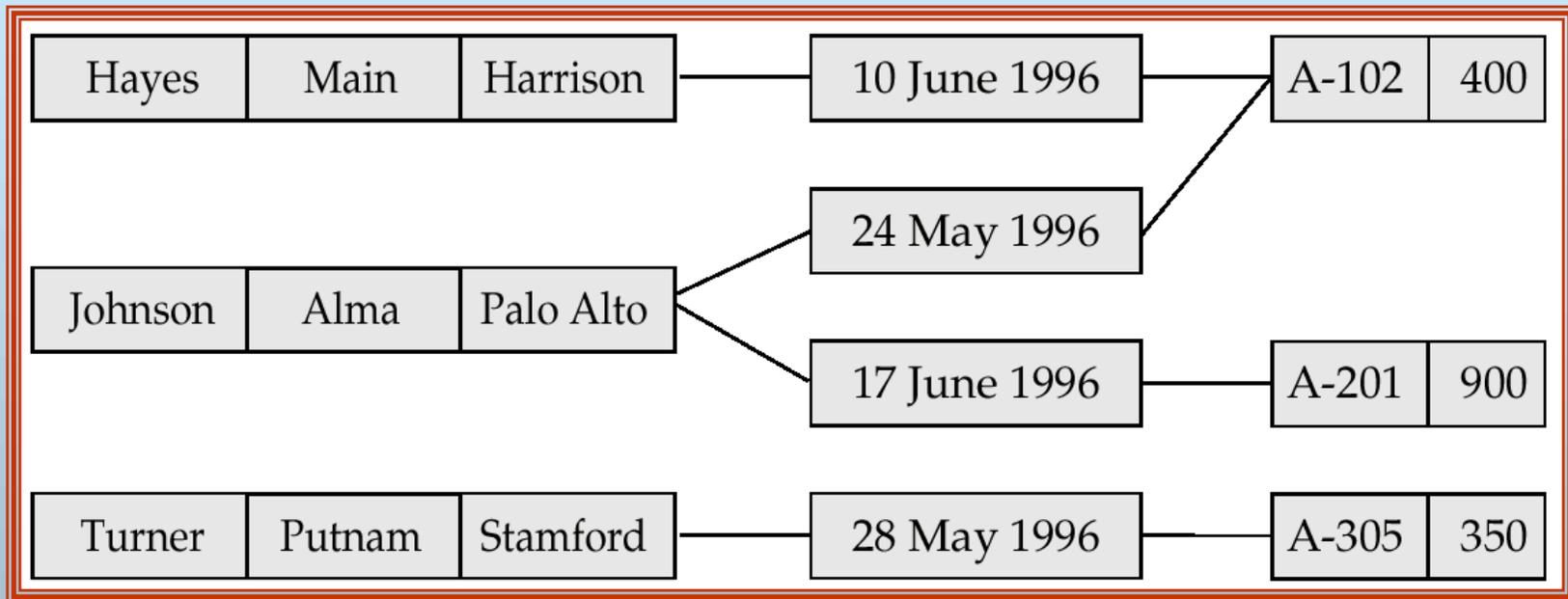
Sample Database Corresponding to Diagram of Figure A.8b
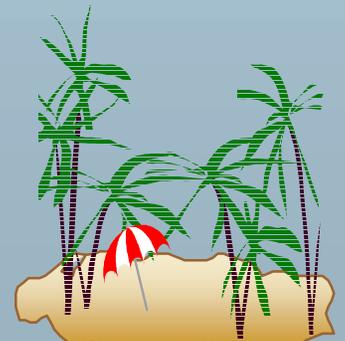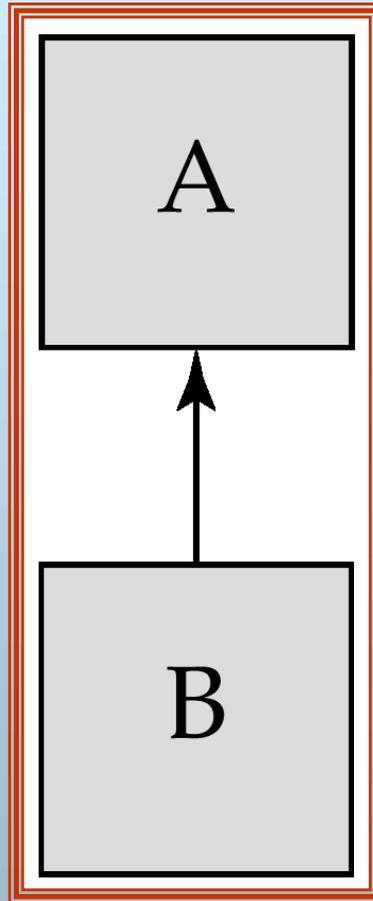
Two Data-Structure Diagrams

customer-name	customer-street	customer-city		account-number	balance

customer   account

(a)

customer-name	customer-street	customer-city	account-number	balance

customer   account

access-date

access-date

(b)
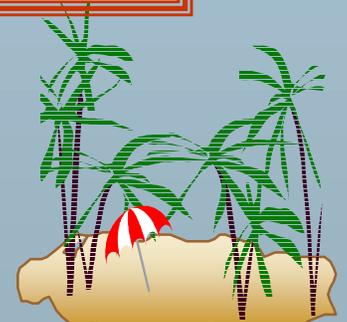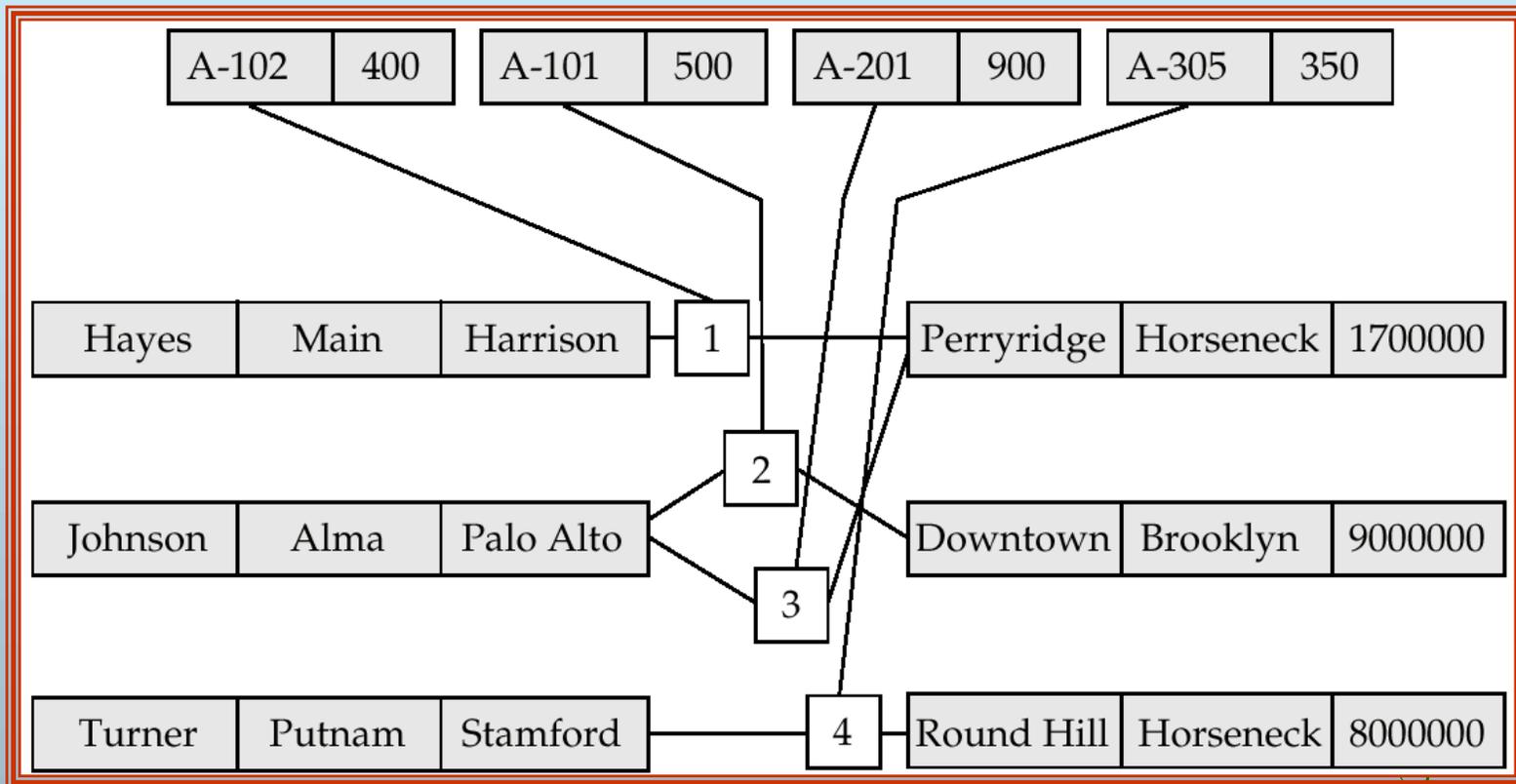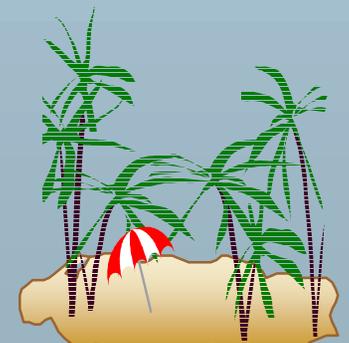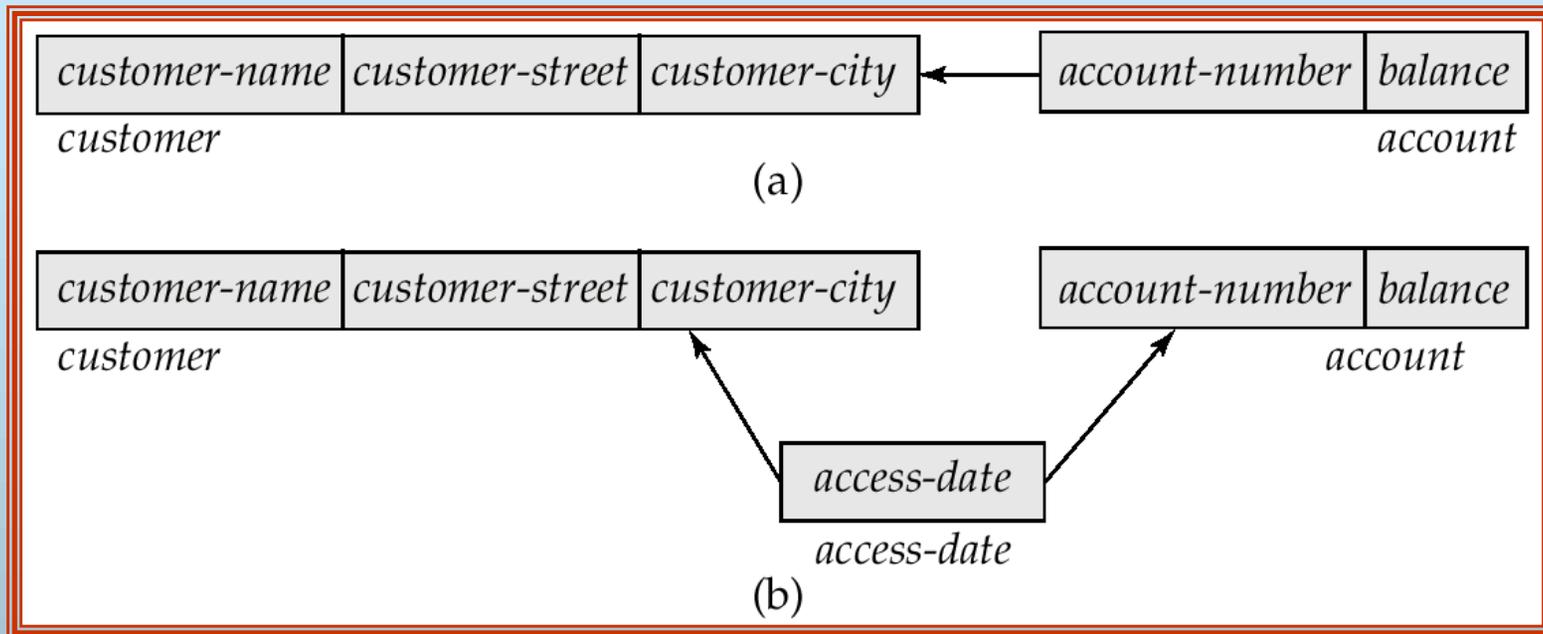
Sample Database Corresponding to the Diagram of Figure A.11
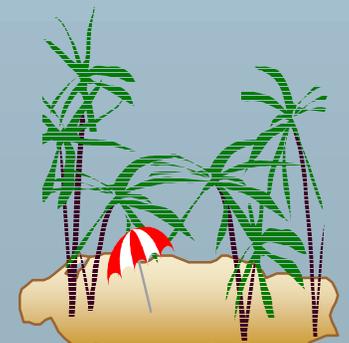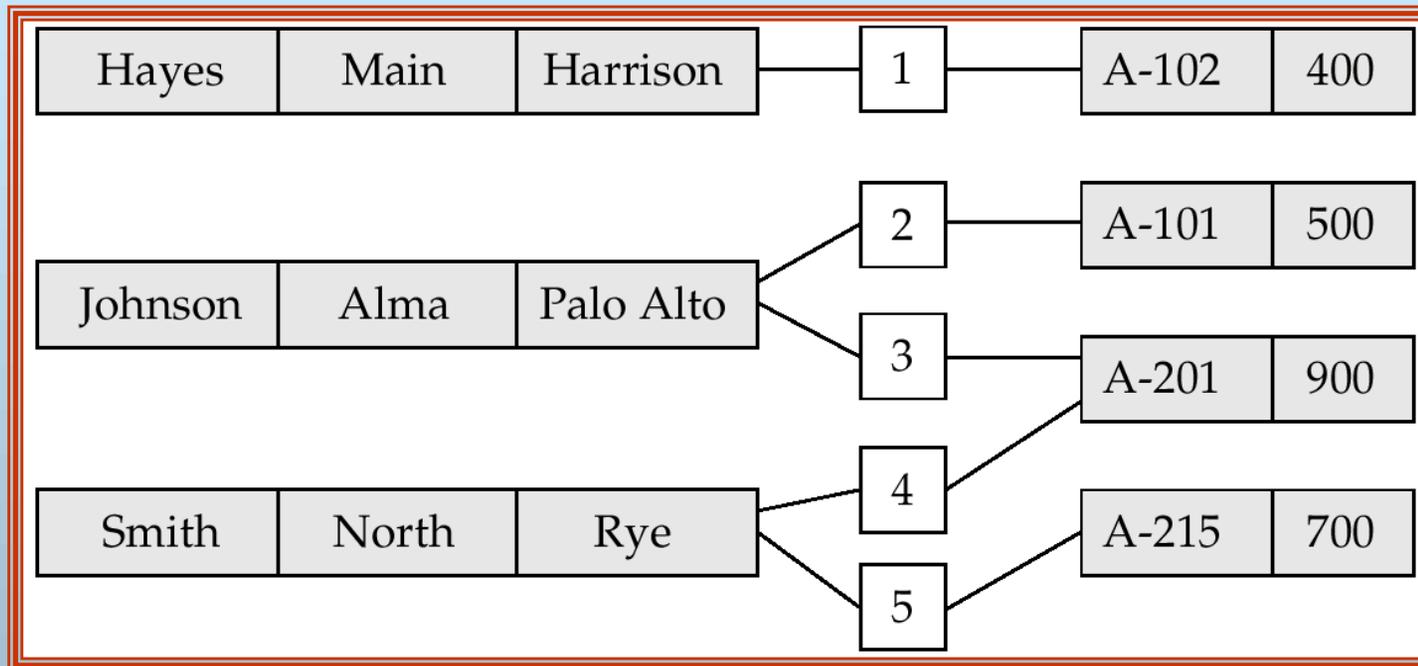
DBTG Set

Three Set Occurrences
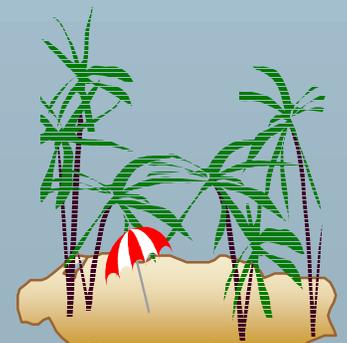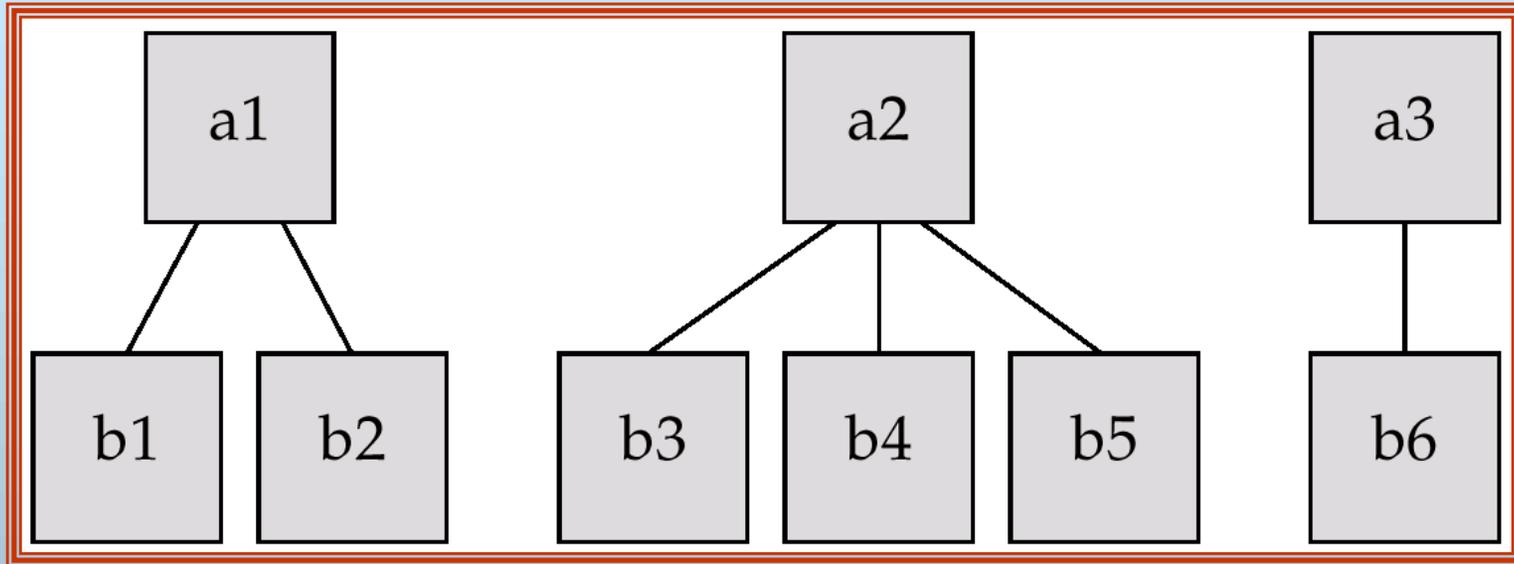
Data-Structure and E-R Diagram
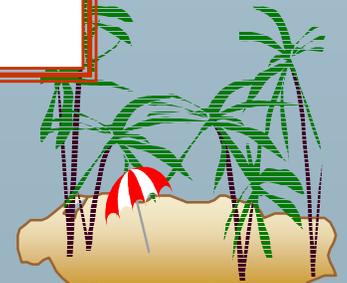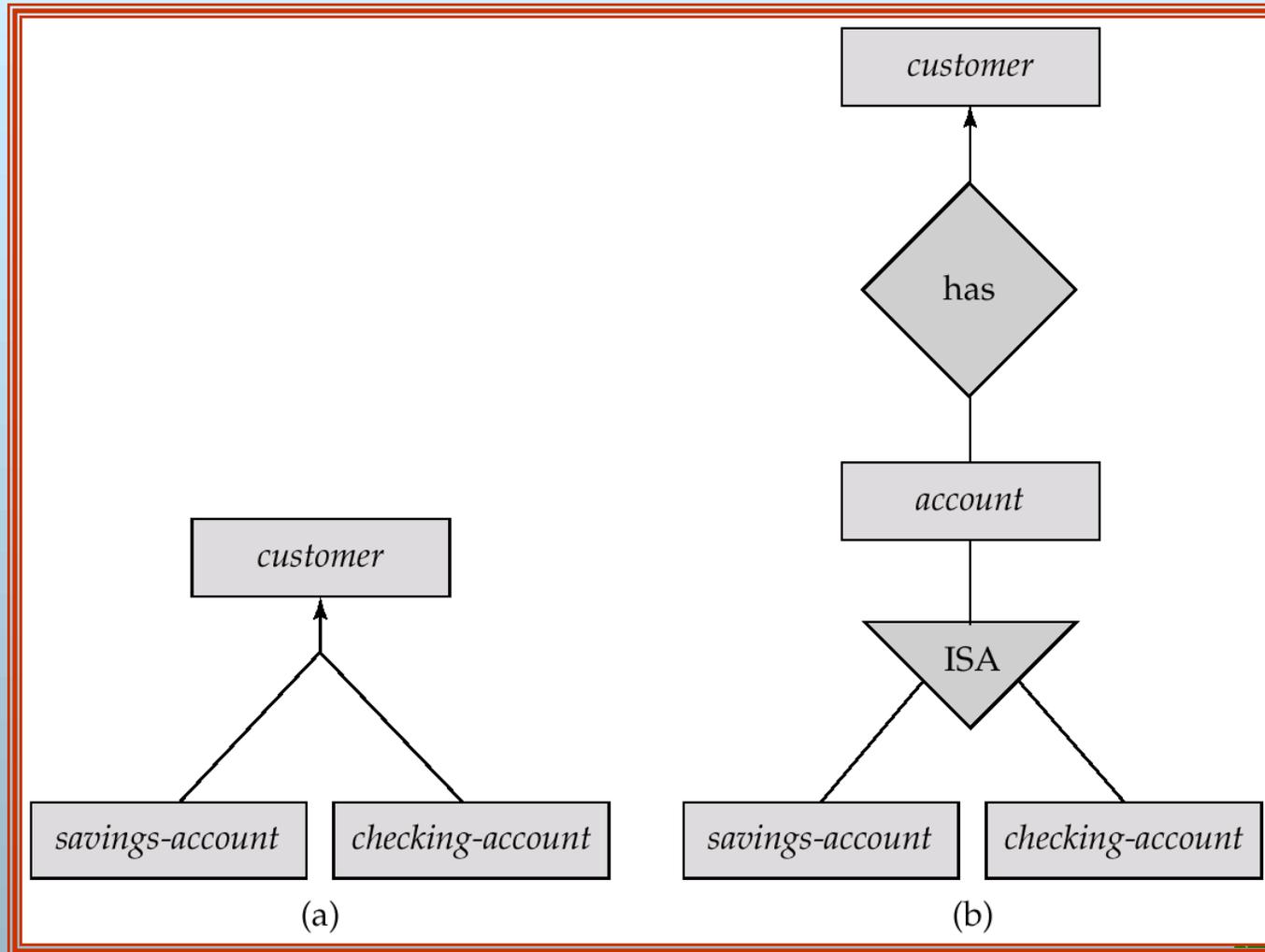
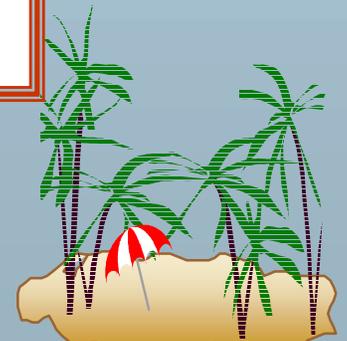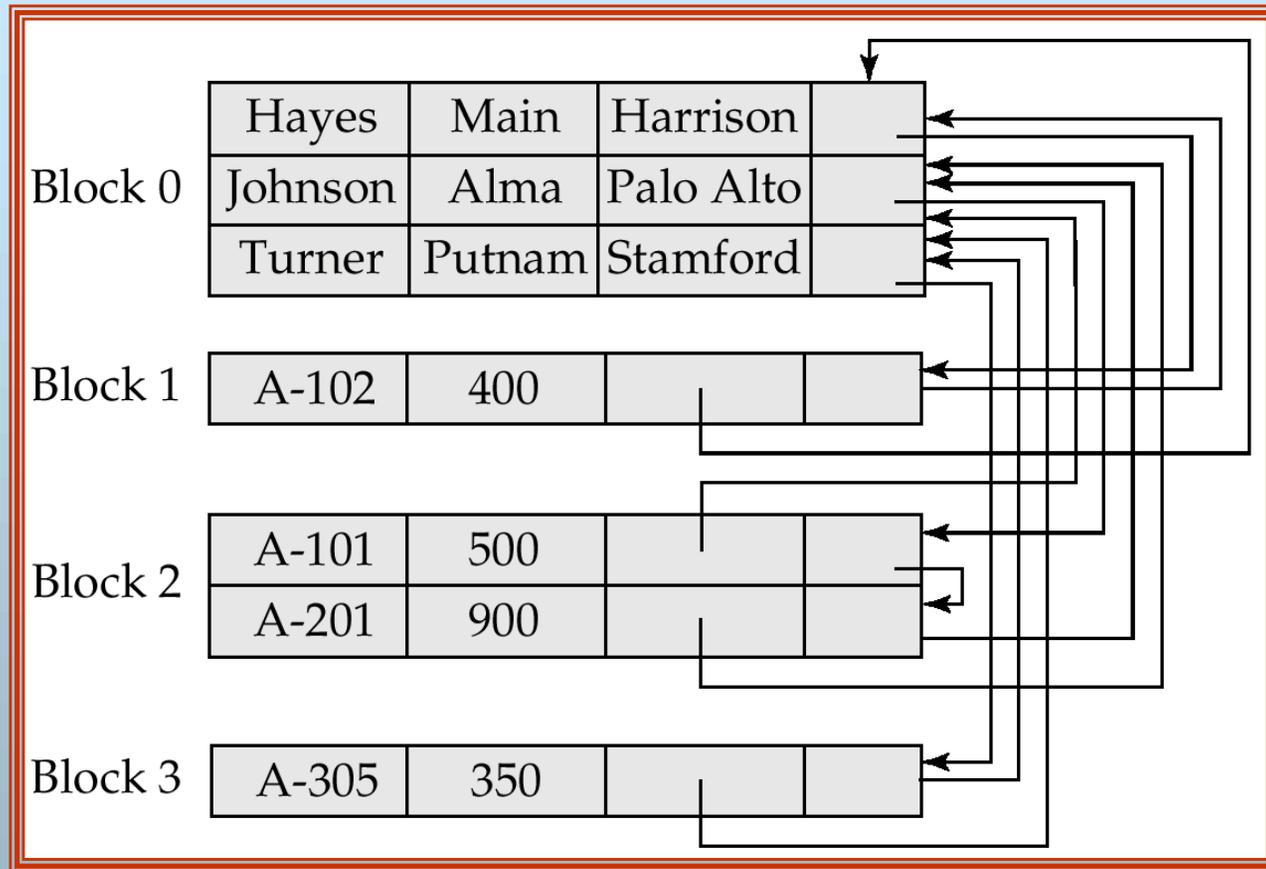# A *customer* Record

Turner	Putnam	Stamford
	Field	Horseneck
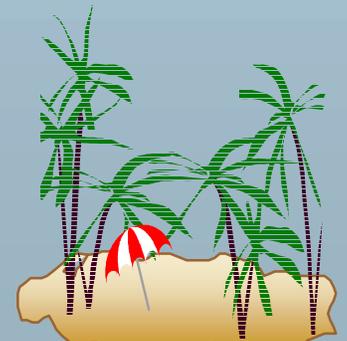
Clustered Record Placement for Instance for Figure A.1

Hayes	Main	Harrison	
Johnson	Alma	Palo Alto	
Turner	Putnam	Stamford	

Block 0

A-102	400		

Block 1

A-101	500		
A-201	900		

Block 2

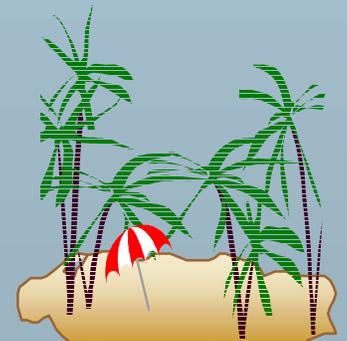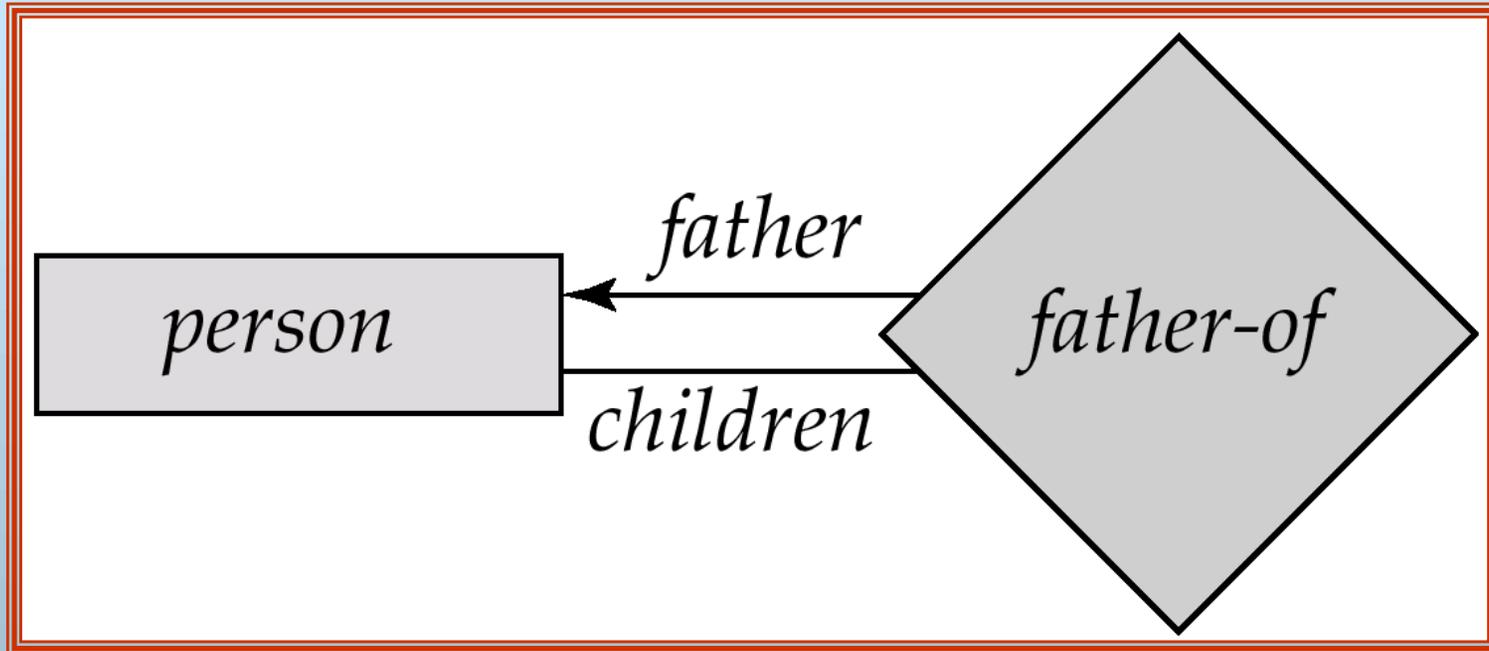|			
A-305	350		

Block 3

Class Enrollment E-R Diagram
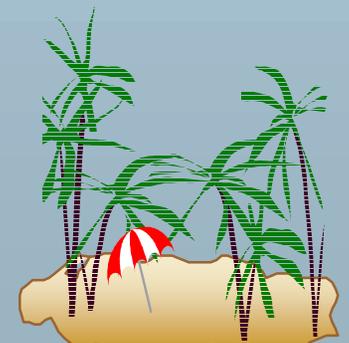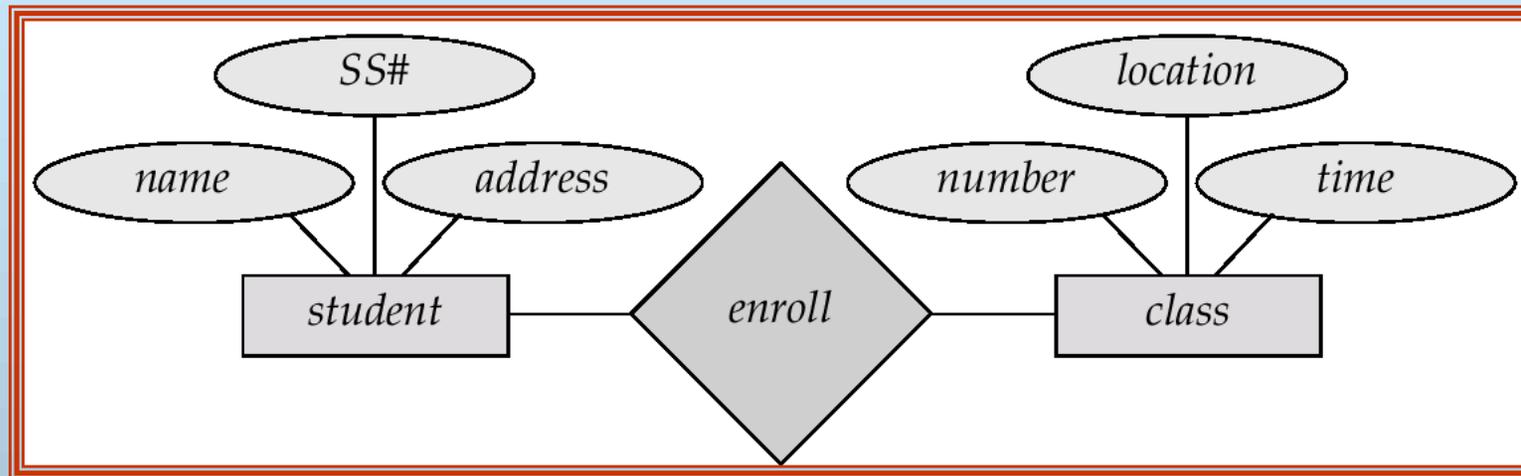
Parent—Child E-R Diagram

Car-Insurance E-R Diagram