DEPARTMENT OF COMMERCE (CA)
DATABASE MANAGEMENT SYSTEM (Semester-III)
II B.COM(CA) Sub Code-18BCA32C
UNIT - V

Network Approach : Architecture of DBTG System. DBTG Data Structure : The set construct, Singular sets, Sample Schema, the external level of DBTG – DBTG Data Manipulation. Disaster Management recovery system

Basic concepts:

A network database consists of a collection of records, which are connected to one another through links. A record is in many respects similar to an entity in the entity-relationship model. Each record is a collection of fields (attributes), each of which contains only one value. A link can be viewed as a restricted (binary) form of relationship in the sense of the E-R model.
To illustrate, consider a database representing a customer-account relationship in a banking system. There are two record types, customer and account. As we saw earlier, the customer record type can be defined, using Pascal-like notation, as follows:

```
type customer = record
                    name: string;
                    street: string;
                    city: string;
                end
```

The account record type can be defined as follows:

```
type account = record
                    number: integer;
                    balance: integer;
                    end
```

The sample database in figure A.1 shows that Lowman has account 305, Camp has accounts 226 and 177, and kahn has account 155.

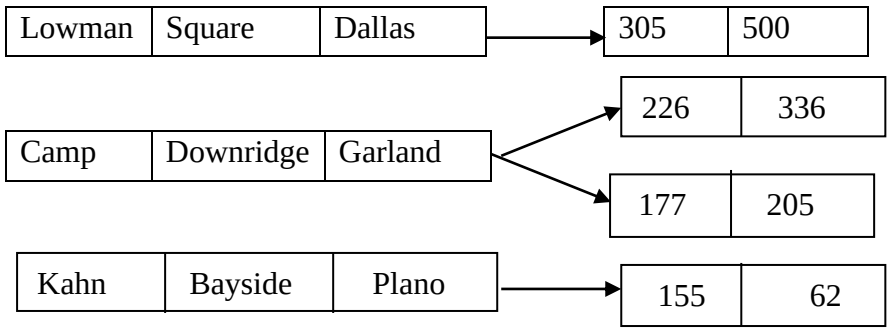


Fig:1
Sample database

Data-structure diagrams: [Architecture of network model]

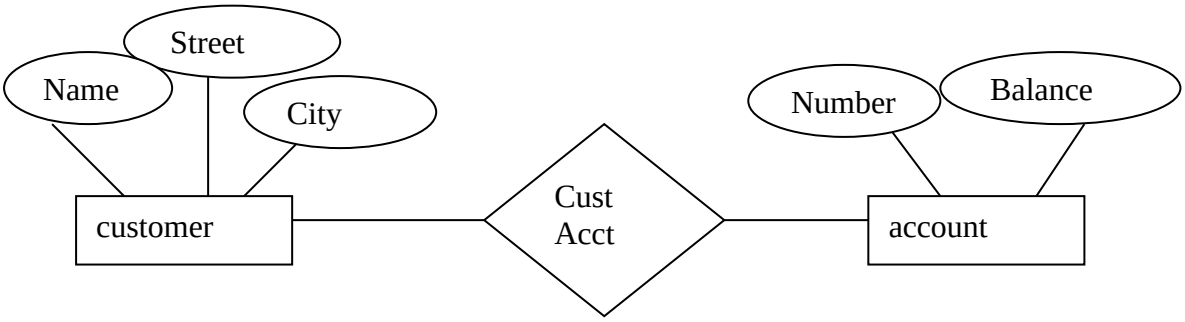
A data-structure diagram is the scheme representing the design of a network database. Such a diagram consists of two basic components:

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

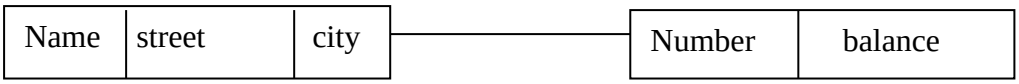
A data-structure diagram serves the same purpose as an entity-relationship diagram; namely, it specifies the overall logical structure of the database. We shall consider the representation of binary, ternary etc. relationships of entity-relationship diagrams.

BINARY RELATIONSHIP

The entity-relationship diagram for banking example is shown as follows:



E-R diagram (a)



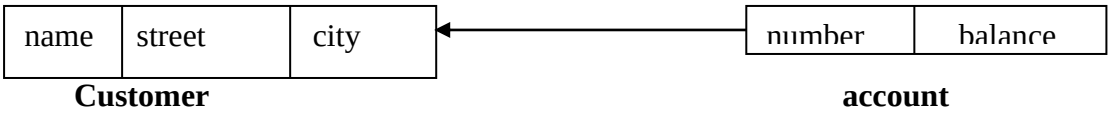
(b)

FIG:2

The above shown diagram (a) is the entity-relationship diagram and consists of two entity-sets customer and account, and they are related through a binary 'many-to-many' relationship 'custacct' with no descriptive attributes.

The diagram shows that a customer may have several accounts and that an account may belong to several different customers. The corresponding data-structure diagram is shown in figure (b). Here the record type customer corresponds to the entity set customer. It includes three fields-name, street and city.

Similarly, account is the record type corresponding to account entity-set and includes the attributes number and balance. Since, in the E-R diagram of above figure the CustAcct relationship is many-to-many, we draw no arrows on the link CustAcct diagram. If the relationship custacct were one-to-many from customer to account then the link custacct would have an arrow pointing to customer record type. The representation is shown as follows:



A sample database corresponding to the data-structure diagram of figure as shown. Since the relation is many-to-many, we show that katz has accounts 256 and 347 and that account 347 is owned by katz and Doner. A sample database corresponding to the data-structure diagram is shown here:

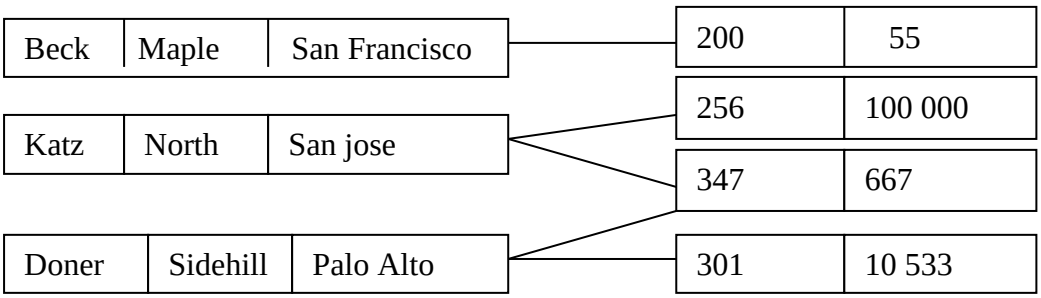


Fig:4
Sample database corresponding t diagram of FIG:3a

Since the relationship is one-to-many
From customer to account, a customer may have more than one account, as is the case with Camp, who owns both 226 and 177. An account, however, cannot belong to more than one customer, as is indeed observed in the sample database.
Finally, a sample database corresponding to the data-structure diagram of fig:3b is shown in the FIG:1.

How to replace the E-R diagram shown in FIG:2a if the descriptive attribute has to be included?

The transformation is more complicated because the link cannot contain any data value.So new record type has to be created and links need to be established as follows:
If for example we consider the E-R diagram shown in FIG:2a and we are trying to add the descriptive attribute date to the custacct relationship to denote the last time the customer has accessed the account.The newly derived E-R diagram is shown here

To transform this diagram to a data-structure diagram we need to:
1:Replace entities customer and account with record types customer and account
2:Create a new record type date with a single field to represent the date.
3:Create the following many-to-one links:

*custdate from the date record type to the customer record type
*acctdate from the date record type to the account recotd type

The DBTG CODASYL Model

The Database Task Group wrote the first database standard specification, called the CODASYL DBTG 1971 report, in the late 1960s. Then a number of changes have been suggested to that report, the last official one in 1978.The rules or standards advised by DBTG group are

Link restriction
DBTG Sets
Repeating Groups

Link Restriction

In the DBTG model, only many-to-one links can be used. Many-to-many links are disallowed in order to simplify the implementation. One-to-one links are represented using a many-to-one link. Let us illustrate this with the help of an example:

Consider a binary relationship that is either one-to-many or one-to-one. If for our customer-account database, if the custacct relationship is one-to-many with no descriptive attributes and with descriptive attribute is shown in the following figure:

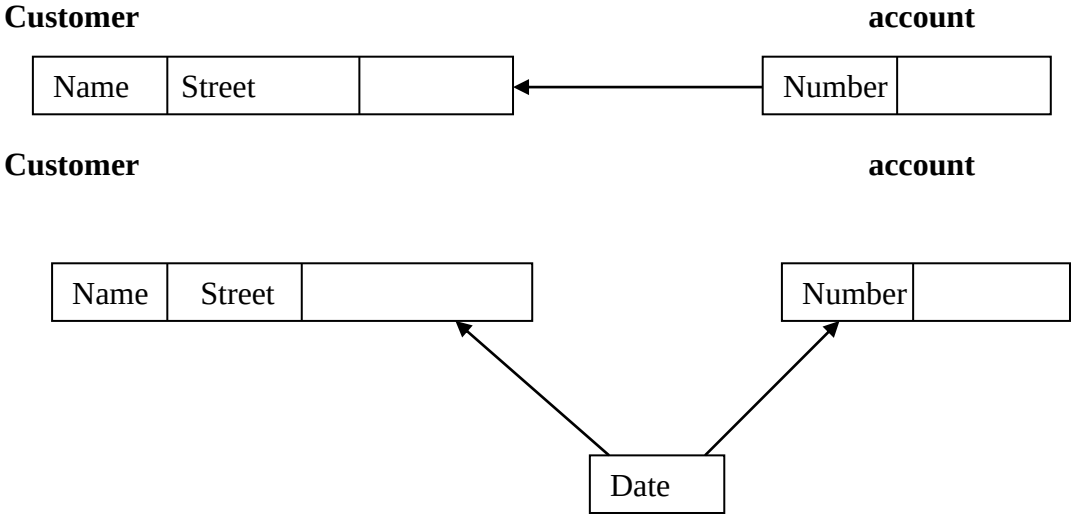
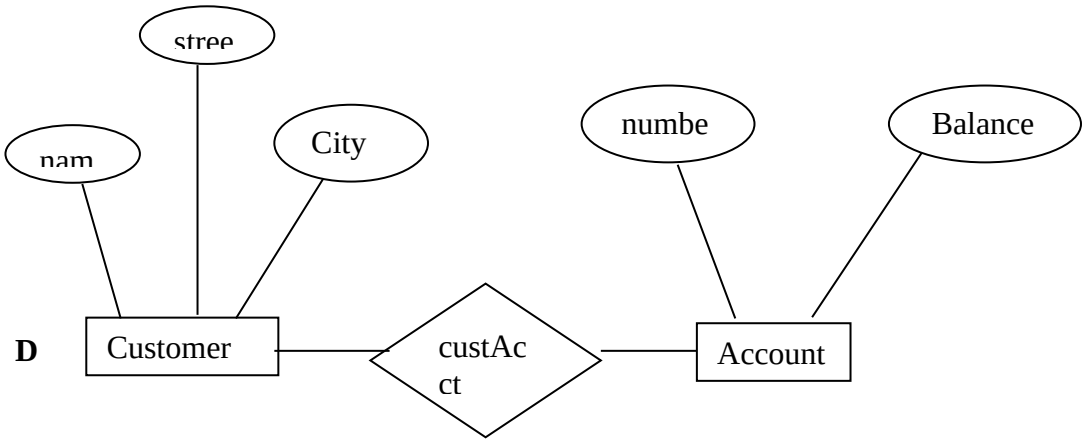


Fig: Two data-structure diagrams

If the custacct relationship is many-to-many then our transformation algorithm must be refined as follows. If the relationships have no descriptive attributes then the following algorithm must be employed:

1:Replace the entity sets customer and account with record types customer and account.
2:Create a new dummy record type Rlink that may either have no fields or have a single field containing an externally defined unique identifier.
3:Create the following two many-to-one links:
custrlink from rlink record type to customer record type
*acctlink from record type to account record type.



DBTG sets

Given that only many-to-one links can be used in the DBTG model, a data-structure diagram consisting of two record types that are linked together has the general form of the following figure:

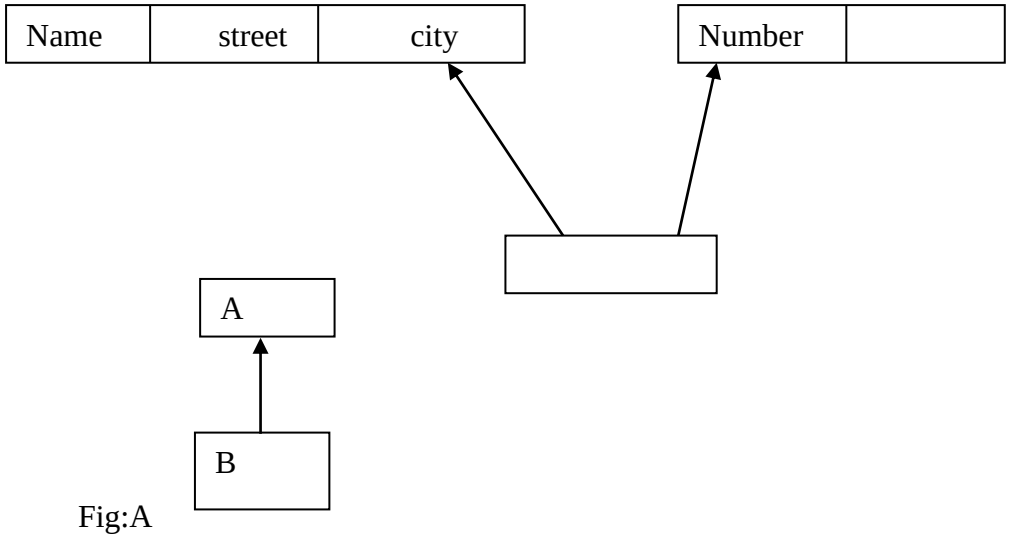


Fig:A

The above shown structure is referred in the DBTG model as a DBTG-set. The name of the set is usually chosen to be the same as the name of the link connecting the two record types.

In each such DBTG-set, the record type A is said as the owner (or parent) of the set, and the record type B is said as the member (or child) of the set. Each DBTG-set can have any number of set occurrences-that is actual instances of linked records.

For example in the figure we are having three occurrences corresponding to the DBTG-set of figure A.

Since many-to-many links are disallowed, each set occurrence has precisely one owner and zero or more member records. In addition, no member record of a set can participate. Simultaneoulsy in several set occurrences of different DBTG-sets.

To illustrate, consider the data-structure diagram shown here. There are two DBTG-sets.

- Custacct, having customer as the owner of the DBTG-set, and account as the member of the DBTG-set.
- Brncacct, having branch as the owner of the DBTG-set, and account as the member of the DBTG-set.

- The set custacct may be defined as follows:

Set name is custacct
Owner is customer
Member is account

The set brncacct may be defined similarly as
Set name is brncacct
Owner is branch
Member is account

An instance of the database is shown here:

Five set occurences are shown: three of set custacct,and two of set brncacct

1:owneer is customer record Lowman with a singke member account record 305
2:owner is customer record Camp with two member account records 177 and 226
3:Owner is cuatomer record Kahn with three member account records 155,402 and 408.
4:Owner is branch record Hillside with three member account records 305,226 and 155.
5:Owner is branch record Valleyview with three member account records 177,402 and 408

Here the fact, an account record cannot appear in more than one set occurrence of one individual set type. This is because an account can belong to exactly one customer, and can be associated with only one bank branch. An account can appear in two set occurrences of different set types. For example, acccount 305 is a member of set occurrence 1 of type custacct and is also a member of set occurrence 4 of type brncacct.

The member records of a set occurrence may be ordered in a variety of ways.

Repeating Groups:

The DBTG model provides a mechanism for a field to have a set of values, rather than one single value.
For example, Suppose that a customer have several addresses. In this case, the customer record type will have the (street, city) pair of fields is defined as repeating group. So the customer record for Kahn is shown here:

The repeating groups construct is another way of representing the notion of weak entities in the E-R model. To illustrate we shall split the entity set customer into two sets:

*Customer, with descriptive attribute name
*Address, with descriptive attribute street and city.

The address entity set is weak entity set, since it depends on the strong entity set customer.

DBTG data retrieval facility

The data manipulation language of the DBTG proposal consists of a number of commands that are embedded in a host language. The commands are explained as follows:

The Find and Get commands

The two most frequently used DBTG commands are

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

Access of individual records:

The find command has a number of forms. There are two different find commands for locating individual records in the database. the simplest command has the form:

Find any <record type> using <record-field>

Purpose: 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. The following currency pointers are set to point to that record:

*The currency 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

For example: Construct the DBTG query that prints the street address of Lowman.

Customer. name:=”Lowman”;
Find any customer-using name;

Get customer;
Print (customer.street);

To display the duplicate records the command is

Find duplicate <record type> using <record-field>

Which locates the next record, which matches the <record-field>.

Example: Construct the DBTG-query that prints the names of all the customers who live in Dallas:

```
Customer.city:=”Dallas”;
Find any customer-using city;
While DB-status = 0 do
Begin
        Get customer;
        Print(customer.name);
        Find duplicate customer using city;
End;
```

Access of records within a set

Purpose: Locate records in a particular DBTG-set.

There are three different types of commands.

The basic find command is

Find first <record type> within <set-type>

Which locates the first database record of type <record type> belonging to the current <set-type>.

To locate the other members of a set the command is

Find next <record-type> within <set-type>

This command finds the next elements in the set <set-type>

Example: Construct the DBTG query that prints the total balance of all accounts belonging to Lowman.

```
Sum: =0;
Customer. name:=”Lowman”;
Find any customer-using name;
Find first account within custacct;
While DB-status =0 do
Begin
        Get account;
        Sum:=sum + account. Balance;
        Find next account within custacct;
End
Print (sum);
```

To find the owner of a particular DBTG-set .The command used is

Find owner within <set-type>

Example: Construct the DBTG-query that prints all the customers of the Hillside branch:

```
Branch-name:=”Hillside”;
Find any branch-using name;
Find first account within brncacct;
While DB-status=0 do
Begin
        Find owner within custacct;
        Get customer;
        Print(customer. name);
        Find next account within brncacct;
End
```

DBTG update facility

Creating new records

To create a new record of type <record type> we insert the appropriate values in the corresponding <record type> template. And the command used is

Store <record type>

Example: Construct the DBTG query to add a new customer Jackson to the database.

```
Customer.name:=”Jackson”;
Customer.street:=”Old road”;
Customer.city:=”Richardson”;
Store customer;
```

Modifying an existing record

In order to modify an existing record of type <record type> we must find the record in the database, get that record into the memory, and then change the desired fields in the template of <record type>. Once this is accomplished, we reflect the changes to the record to which the currency pointer of <record type> points by executing the command:

Modify <record type>

The DBTG model requires the find command to be executed prior to modifying a record must have the additional clause “for update” so that the system is aware of the fact that the record is to be modified.

Example:
Construct the DBTG program to change the street address of Kahn to North Loop.

```
Customer.name:=”Kahn”;
Find for update any customer using name;
Get customer;
Customer.city:=”North Loop”;
```

Modify customer;

Deleting a record

To delete an existing record of type <record type> we use the command:

Erase <record type>

Example:
The query to construct the DBTG program to delete account 402 belonging to Kahn:

```
Finish:=false;
Customer.name:=”Kahn”;
Find any customer using name;
Find for update first account within custacct;
While DB-status=0 and not finish do
Begin
        Get account;
        If account.number =402 then
        Begin
        Erase account;
        Finish: = true;
        End;
        Else
        Find for update next account within custAcct
End;
```

It is possible to delete an entire set occurrence by finding the owner of the set – say, a record of type <record type> - and executing.

Erase all<record-type>

This will delete the owner of the set as well as its entire member. If a member of the set is an owner of another set the members of that set are also deleted. That the erase all operation is recursive.

Eg.

Consider the DBTG program to delete customer “Camp” and all of her accounts.

```
Customer.name :=”Camp”;
Find for update any customer using name;
Erase all customer.
```

DBTG set-processing facility

This mainly concerns with the mechanism of inserting records into and removing records from a particular set occurrence.

The connect statement

To insert a new record of type <record type> into a particular occurrence of <set-type> we must first insert the record into the database, then set the currency pointers of <record type> and <set type> to point to the appropriate record and set occurrence.

The command used is

Connect <record type> to <set-type>

A new record can be inserted as follows:
1:create a new record of type **<record type> .**
2:Find the appropriate owner of the set <set type>.
3:Insert the new record into the set by executing the connect statement.

Example:

Create the DBTG query for creating new account 267 which belongs to Jackson:

```
Account.number:=267;
Account.balance:=0;
Store account;
Customer.name:=”Jackson”;
Find any customer using name;
Connect account to custacct;
```

The Disconnect statement

In order to remove a record of type <record type> from a set occurrence of <set-type>, we need to set the currency pointer of <record type> and <set-type> to point to the appropriate record and set occurrence. Once this is accomplished, the record can be removed from the set by executing

Disconnect <record-type> from <set-type>

Eg. To remove account 177 from the set occurrence of type custacct.

```
Account.number :=177;
Find for update any account using number;
Get account;
Find owner within custacct;
Disconnect account from custacct;
```

The reconnect statement

In order to move a record of type <record-type> from one set occurrence to another set occurrence of type <set-type>, we need to find the appropriate record and the owner of the set occurrence to which the record is to be moved. Once this is done, we can move the record by executing:

Reconnect <record-type> to <set-type>

Consider the DBTG program to move all accounts of Lowman that are currently at the hillside branch to the valley view branch.

```
Customer.name :=”Lowman”;
Find any customer-using name;
Find first account within custacct;
While DB-status =0 do
    Begin
        Find owner within brncacct;
        Getbranch;
        If branch.name = “hillside”  then
        Begin
            Branch.name:=”Valley view”;
            Find any branch-using name;
            Reconnect account to brncacct;
        End;
        Find next account within custacct;
    End;
```

Set Insertion and Retention

When a new set is defined, we must specify how member records are to be inserted. In addition, we must specify the conditions under which a record must be retained in the set occurrence in which it was initially inserted.

Set Insertion

A newly created record of type <record type > of a set type <set type > can be added to a set occurrence either explicitly (MANUALLY) or implicitly (automatically). This distinction is specified at set definition time via

Insertion is < insert mode >

Where < insert mode > can take one of two forms.

♣ Manual : The new record can be inserted into the set manually (explicitly) by executing .

Connect < record type > to <set-type>

♣Automatic : The new record is inserted into the set automatically (implicitly) when it is created , that is , when we execute .

Store < record type >

In either case, just prior to insertion, the <set-type> currency pointer must point to the set occurrence into which the insertion is to be made.

Set Retention

There are various restrictions on how and when a member record can be removed from a set occurrence into which it has been inserted previously. These restrictions are specified at set definition time via

Retention is < retention-mode >

Where <retention-mode> can take one of the three forms

♣Fixed : Once a member record has been inserted into a particular set occurrence , it cannot be removed from that set . If retention is fixed , then to reconnect a record to another set , we must first erase that record , re-create it , and then insert it into the new set occurrence .

♣Mandatory : Once a member record has been inserted into a particular set occurrence , it can be reconnected only to another set occurrence of type <set-type>. It can neither be disconnected nor be reconnected to a set of another type .

♣Optional : No restrictions are placed on how and when a member record can be reconnected , disconnected ,and connected at will .The decision as to which to option to choose is dependent on the application .

Deletion

When a record is deleted (erased) and that record is the owner of set occurrence of type <set-type> , the best way of handling this deletion depends on the specification of the set retention of <set-type>

♣ If the retention status is optional, then the record will be deleted and every member of the set it owns will be disconnected. These records, however, are kept in the database.

♣ If the retention status is fixed, then the record and all of its owned members will be deleted. This follows from the fact that the fixed status indicates that a member record cannot be removed from the set occurrence without being deleted.

♣If the retention status is mandatory, then the record cannot be erased this is because the mandatory status indicates that a member record must belong to a set occurrence; it cannot be disconnected form that set.

Set Ordering

The members of a set occurrence of <set-type> may be ordered in a variety of ways. A programmer specifies these orders when the set is defined

Order is <order-mode>

Where <order-mode> can be

♣ First : When a new record is added to a set , it is inserted in the first positive . Thus, the set is in reverse chronological ordering

♣ Last : When a new record is added to a set , it is inserted in the ;last position . Thus, the set is in chronological ordering

♣ Next : Suppose that the currency pointer of <set-type> points to record X . if X is a member type , then when a new record is added to the set . It is inserted in the position following X. If X is an owner type, then when a new record is added, it is inserted in the last position.

♣ Prior : Suppose that the currency pointer of ,set-type> points to record X . If X is a member type, then when a new record is added to the set it is inserted in the position just prior to X. If X is an owner type, then when a new record is added, it is inserted in the last position.

♣ System default : When a new record is added to a set , it is inserted in an arbitrary position determined by the system .

♣ Sorted : When a new record is added to a set , it 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. The programmer must specify whether members are ordered in ascending or descending order relative to that key.

Database systems, like any other computer system, are subject to failures but the data stored in it must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transactions are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database.

There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crash, transaction errors, viruses, catastrophic failure, incorrect commands execution etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used.

Disaster Management recovery system

Recovery techniques are heavily dependent upon the existence of a special file known as a **system log**. It contains information about the start and end of each transaction and any updates which occur in the **transaction**. The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.

- The log is kept on disk start_transaction(T): This log entry records that transaction T starts the execution.
- read_item(T, X): This log entry records that transaction T reads the value of database item X.
- write_item(T, X, old_value, new_value): This log entry records that transaction T changes the value of the database item X from old_value to new_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.
- commit(T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.
- abort(T): This records that transaction T has been aborted.
- checkpoint: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

A transaction T reaches its **commit** point when all its operations that access the database have been executed successfully i.e. the transaction has reached the point at which it will not **abort** (terminate without completing). Once committed, the transaction is

permanently recorded in the database. Commitment always involves writing a commit entry to the log and writing the log to disk. At the time of a system crash, item is searched back in the log for all transactions T that have written a start_transaction(T) entry into the log but have not written a commit(T) entry yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process

- **Undoing –** If a transaction crashes, then the recovery manager may undo transactions i.e. reverse the operations of a transaction. This involves examining a transaction for the log entry write_item(T, x, old_value, new_value) and setting the value of item x in the database to old-value.There are two major techniques for recovery from non-catastrophic transaction failures: deferred updates and immediate updates.
- **Deferred update –** This technique does not physically update the database on disk until a transaction has reached its commit point. Before reaching commit, all transaction updates are recorded in the local transaction workspace. If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed. It may be necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database. Hence, a deferred update is also known as the **No-undo/redo algorithm**
- **Immediate update –** In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are recorded in a log on disk before they are applied to the database, making recovery still possible. If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back hence we require both undo and redo. This technique is known as **undo/redo algorithm.**
- **Caching/Buffering –** In this one or more disk pages that include data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk. A collection of in-memory buffers called the DBMS cache is kept under control of DBMS for holding these buffers. A directory is used to keep track of which database items are in the buffer. A dirty bit is associated with each buffer, which is 0 if the buffer is not modified else 1 if modified.
- **Shadow paging –** It provides atomicity and durability. A directory with n entries is constructed, where the ith entry points to the ith database page on the link. When a transaction began executing the current directory is copied into a shadow directory. When a page is to be modified, a shadow page is allocated in which changes are made and when it is ready to become durable, all pages that refer to original are updated to refer new replacement page.

Some of the backup techniques are as follows :

- **Full database backup –** In this full database including data and database, Meta information needed to restore the whole database, including full-text catalogs are backed up in a predefined time series.
- **Differential backup –** It stores only the data changes that have occurred since last full database backup. When same data has changed many times since last full database backup, a differential backup stores the most recent version of changed data. For this first, we need to restore a full database backup.

- **Transaction log backup –** In this, all events that have occurred in the database, like a record of every single statement executed is backed up. It is the backup of transaction log entries and contains all transaction that had happened to the database. Through this, the database can be recovered to a specific point in time. It is even possible to perform a backup from a transaction log if the data files are destroyed and not even a single committed transaction is lost.

REFERENCE:

1. Database systems concepts by Abraham Silberschatz, Henry F. Korth.
2. An Introduction to Database System – C. Dsai.
3. An introduction to Database Systems (Seventh Edition) – C.J. Date
4.www.geeksforgeeks.com

Prepared by Dr.N.SHANMUGAVADIVU