II B.Sc Computer Science
18BCS42C – DATABASE MANAGEMENT SYSTEM
UNIT-II

DATA MODELING: Introduction – Data associations – Entities, attributes, relationships – Constraints - Design of Entity Relationship data models (ERD) – Generalization – Aggregation – Conversion of ERD into tables – Introduction to Network data model and Hierarchical data model.

FILE ORGANIZATION: Storage device characteristics – Constituents of a file – Operations on file – Serial files – Sequential files – Index sequential files – Direct files – Binary and Secondary Key Retrieval – Indexing using Tree Structures.

Data Modeling

Introduction

- Data Model is a logical structure of Database.
- It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

Types of Data Models

- There are several types of data models in DBMS.
    - Object based logical Models – Describe data at the conceptual and view levels.
    - Record based logical Models
    - Physical Data Model
- Object based logical Models
    - E-R Model
    - Object oriented Model
- Record based logical Models – Like Object based model, they also describe data at the conceptual and view levels. These models specify logical structure of database with records, fields and attributes.
- Relational Model
- Hierarchical Model
- Network Model – Network Model is same as hierarchical model except that it has graph-like structure rather than a tree-based structure. Unlike hierarchical model, this model allows each record to have more than one parent record.

Physical Data Models – These models describe data at the lowest level of abstraction.

Data Associations

Entity Sets

- A *database* can be modelled as:
  - a collection of entities,
  - relationship among entities.
- An *entity* is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- Entities have *attributes*
  - Example: people have *names* and *addresses*
- An *entity set* is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays
    Entity Sets *customer* and *loan*

Customer-id	**Customer-name**	**Customer-street**	**Customer-city**
321-12-3123	Jones	Main	Harrison
019-28-3746	Smith	North	Rye
677-89-9011	Hayes	Main	Harrison
555-55-5555	Jackson	Dupoint	Woodside
244-66-8800	Curry	North	Rye
963-96-3963	Williams	Nassau	Princeton
335-57-7991	Adams	Spring	Pittsfield

Customer

Loan-number	**Amount**
L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-19	500
L-11	900
L-16	1300

Loan

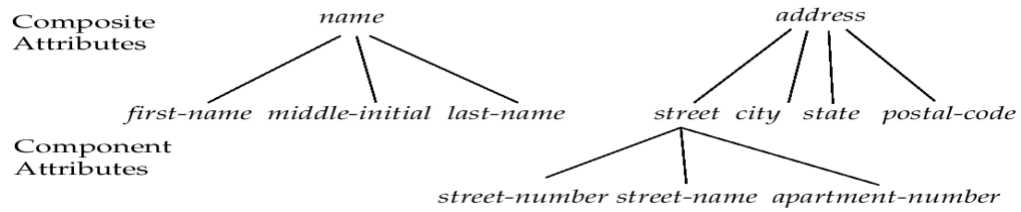
Attributes

- An entity is represented by a set of attributes that has descriptive property possessed by all members of an entity set.
- Example:
  customer = (customer-id, customer-name, customer-street, customer-city)
  loan = (loan-number, amount)
- *Domain*–the set of permitted values for each attribute

Attribute types:

- *Simple* and *composite* attributes.
- *Single-valued* and *multi-valued* attributes
  - E.g. multivalued attribute: *phone-numbers*
- *Derived* attributes
  - Can be computed from other attributes
  - E.g. *age*, given date of birth

Composite Attributes



Relationship Sets

- A relationship is an association among several entities
- Example: Hayes *depositor* A-102
  Customer entity relationship set *account* entity
- A *relationship* set is a mathematical relation among *n*□2 entities, each taken from entity sets{(*e*1, *e*2, … *en*) | *e*1□*E*1, *e*2□*E*2, …, *en*□*En*}where (*e*1, *e*2, …, *en*) is a relationship
- Example:
  (Hayes, A-102) ε *depositor*

Constraints in DBMS

- Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table.
- The whole purpose of constraints is to maintain the **data integrity** during an update/delete/insert into a table.

Types of constraints

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints – PRIMARY KEY, FOREIGN KEY
- Domain constraints
- Mapping constraints

NOT NULL:

- NOT NULL constraint makes sure that a column does not hold NULL value.
- When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default.
- By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.
- Example:

```
CREATE TABLE STUDENT(
```

```
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

UNIQUE:

- UNIQUE Constraint enforces a column or set of columns to have unique values.
- If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

DEFAULT:

- The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35),
PRIMARY KEY (ROLL_NO)
);
```

CHECK:

- This constraint is used for specifying range of values for a particular column of a table.
- When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL CHECK(ROLL_NO>1000),
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35),
PRIMARY KEY (ROLL_NO)
};
```

- In the above example we have set the check constraint on ROLL_NO column of STUDENT table.

- Now, the ROLL_NO field must have the value greater than 1000.

Key constraints:

PRIMARY KEY:

- Primary key uniquely identifies each record in a table.
- It must have unique values and cannot contain nulls.
- In the below example the ROLL_NO field is marked as primary key, that means the ROLL_NO field cannot have duplicate and null values.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
};
```

FOREIGN KEY:

- Foreign keys are the columns of a table that points to the primary key of another table.
- They act as a cross-reference between tables.

Domain constraints:

- Each table has certain set of columns and each column allows a same type of data, based on its data type.
- The column does not accept values of any other data type.
- Domain constraints are **user defined data type** and we can define them like this:
  Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

Entity Relationship Diagram – ER Diagram in DBMS

- An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram).
- An ER model is a design or blueprint of a database that can later be implemented as a database.
- The main components of E-R model are: entity set and relationship set.

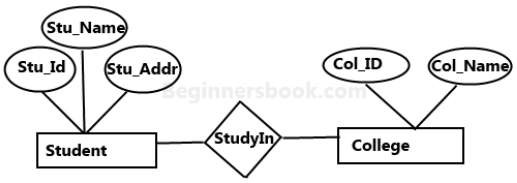
What is an Entity Relationship Diagram (ER Diagram)?

- An ER diagram shows the relationship among entity sets.
- An entity set is a group of similar entities and these entities can have attributes.

- In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.

A simple ER Diagram:

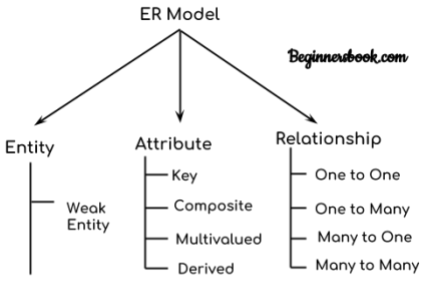
- In the following diagram we have two entities Student and College and their relationship.
- The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time.
- Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.



Sample E-R Diagram

- Here are the geometric shapes and their meaning in an E-R Diagram.
  - Rectangle: Represents Entity sets.
  - Ellipses: Attributes
  - Diamonds: Relationship Set
  - Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
  - Double Ellipses: Multi-valued Attributes
  - Dashed Ellipses: Derived Attributes
  - Double Rectangles: Weak Entity Sets
  - Double Lines: Total participation of an entity in a relationship set

Components of a ER Diagram



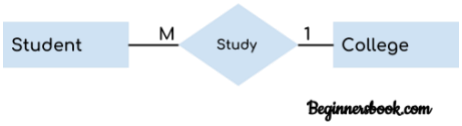
Components of ER Diagram

- As shown in the above diagram, an ER diagram has three main components:

- 1. Entity
- 2. Attribute
- 3. Relationship

1. Entity

- An entity is an object or component of data.
- An entity is represented as rectangle in an ER diagram.
- For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college.



Weak Entity:

- An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity.
- The weak entity is represented by a double rectangle.
- For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.

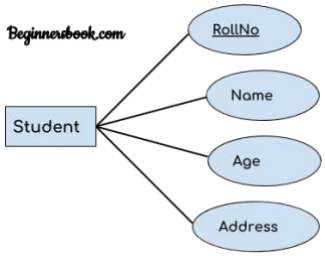
2. Attribute

- An attribute describes the property of an entity.
- An attribute is represented as Oval in an ER diagram.
- There are four types of attributes:
    a. Key attribute
    b. Composite attribute
    c. Multi-valued attribute
    d. Derived attribute

a. Key attribute:

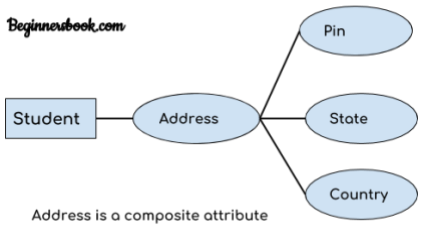
- A key attribute can uniquely identify an entity from an entity set.
- For example, student roll number can uniquely identify a student from a set of students.

- Key attribute is represented by oval same as other attributes however the text of key attribute is underlined.



b. Composite attribute:

- An attribute that is a combination of other attributes is known as composite attribute.
- For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.



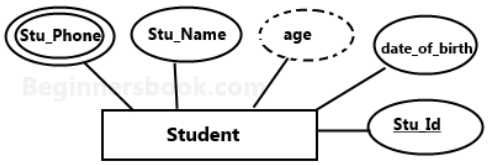
c. Multi-valued attribute:

- An attribute that can hold multiple values is known as multi-valued attribute.
- It is represented with double ovals in an ER Diagram.
- For example – A person can have more than one phone numbers so the phone number attribute is multi-valued.

d. Derived attribute:

- A derived attribute is one whose value is dynamic and derived from another attribute.
- It is represented by dashed oval in an ER Diagram.
- For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multi-valued and derived attributes:

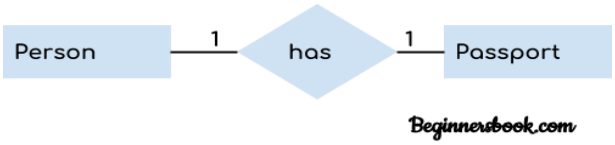


3. Relationship

- A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities.
- There are four types of relationships:
    - One to One
    - One to Many
    - Many to One
    - Many to Many

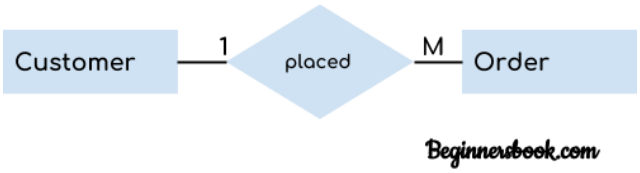
One to One Relationship

- When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship.
- For example, a person has only one passport and a passport is given to one person.



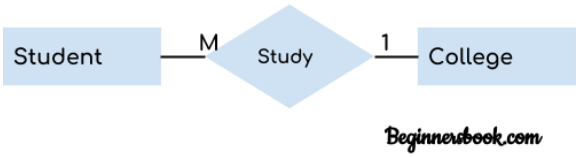
One to Many Relationship

- When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship.
- For example – a customer can place many orders but a order cannot be placed by many customers.



Many to One Relationship

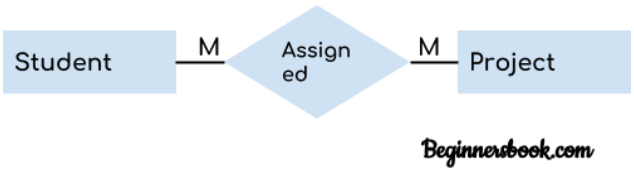
- When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship.
- For example – many students can study in a single college but a student cannot study in many colleges at the same time.



Many to Many Relationship

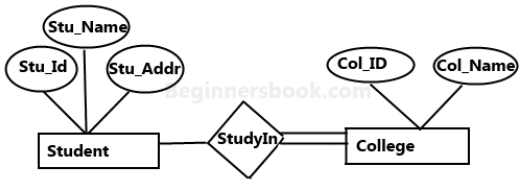
- When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship.

- For example, a can be assigned to many projects and a project can be assigned to many students.



Total Participation of an Entity set

- A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set.
- For example: In the below diagram each college must have at-least one associated Student.

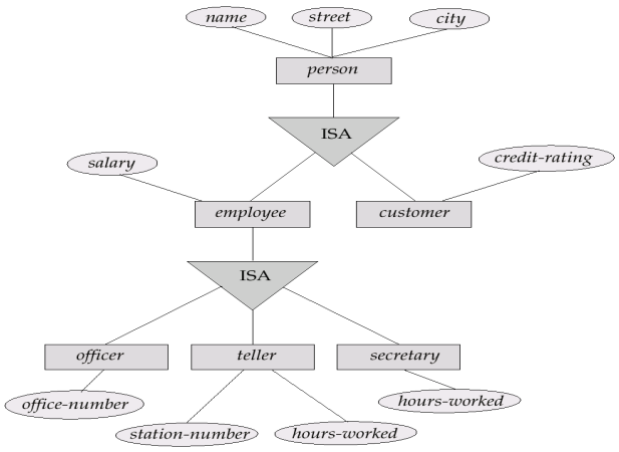


E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

Specialization

- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set.
- These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labelled ISA (E.g. *customer* "is a" *person*).
- Attribute inheritance–a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

Specialization Example



Generalization

- A bottom-up design process – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.
- Can have multiple specializations of an entity set based on different features.
- E.g. *permanent-employee* vs. *temporary-employee*, in addition to *officer* vs. *secretary* vs. *teller*
- Each particular employee would be member of one of *permanent-employee* or *temporary-employee*, and also a member of one of *officer*, *secretary*, or *teller*
- The ISA relationship also referred to as super class -subclass relationship

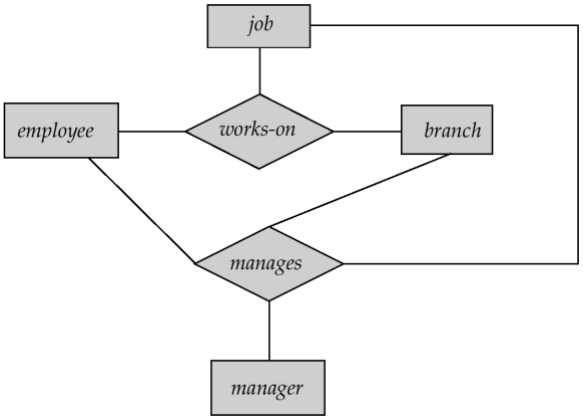
Design Constraints on a Specialization/Generalization

- Constraint on which entities can be members of a given lower-level entity set.
    - condition-defined
        - E.g. all customers over 65 years are members of *senior-citizen* entity set; *senior-citizen* ISA *person*.
    - user-defined
- Constraint on whether or not entities may belong to more than one lower-level entity set within a single generalization.
    - Disjoint
        - An entity can belong to only one lower-level entity set
        - Noted in E-R diagram by writing *disjoint* next to the ISA triangle
    - Overlapping
        - An entity can belong to more than one lower-level entity set.
- Completeness constraint--specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.
    - total: an entity must belong to one of the lower-level entity sets
    - partial: an entity need not belong to one of the lower-level entity sets

Aggregation

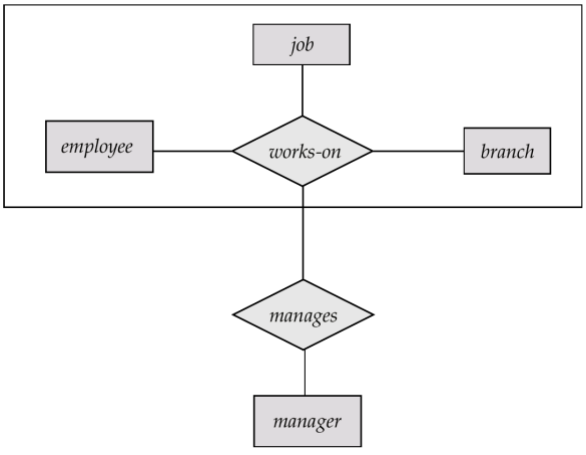
- Consider the ternary relationship *works-on*, which we saw earlier

- Suppose we want to record managers for tasks performed by an employee at a branch



- Relationship sets *works-on* and *manages* represent overlapping information
    - Every *manages* relationship corresponds to a *works-on* relationship
    - However, some *works-on* relationships may not correspond to any *manages* relationships
        - So we can't discard the *works-on* relationship
- Eliminate this redundancy via *aggregation*
    - Treat relationship as an abstract entity
    - Allows relationships between relationships
    - Abstraction of relationship into new entity
- Without introducing redundancy, the following diagram represents:
    - An employee works on a particular job at a particular branch
    - An employee, branch, job combination may have an associated manager

E-R Diagram with Aggregation



Convertion of an E-R Schema to Tables

- Primary keys allow entity sets and relationship sets to be expressed uniformly as *tables* which represent the contents of the database.
- A database which conforms to an E-R diagram can be represented by a collection of tables.
- For each entity set and relationship set there is a unique table which is assigned the name of the corresponding entity set or relationship set.
- Each table has a number of columns (generally corresponding to attributes), which have unique names.
- Converting an E-R diagram to a table format is the basis for deriving a relational database design from an E-R diagram.

Representing Entity Sets as Tables

- A strong entity set reduces to a table with the same attributes.

customer-id	*customer-name*	*customer-street*	*customer-city*
019-28-3746	Smith	North	Rye
182-73-6091	Turner	Putnam	Stamford
192-83-7465	Johnson	Alma	Palo Alto
244-66-8800	Curry	North	Rye
321-12-3123	Jones	Main	Harrison
335-57-7991	Adams	Spring	Pittsfield
336-66-9999	Lindsay	Park	Pittsfield
677-89-9011	Hayes	Main	Harrison
963-96-3963	Williams	Nassau	Princeton

Relational model in DBMS

- In relational model, the data and relationships are represented by a collection of inter-related tables.
- Each table is a group of column and rows, where column represents attribute of an entity and rows represents records.

Sample relationship Model: Student table with 3 columns and four records.

Table: Student

Stu_Id	Stu_Name	Stu_Age
111	Ashish	23
123	Saurav	22
169	Lester	24
234	Lou	26

Table: Course

Stu_Id	Course_Id	Course_Name
111	C01	Science
111	C02	DBMS
169	C22	Java
169	C39	Computer Networks

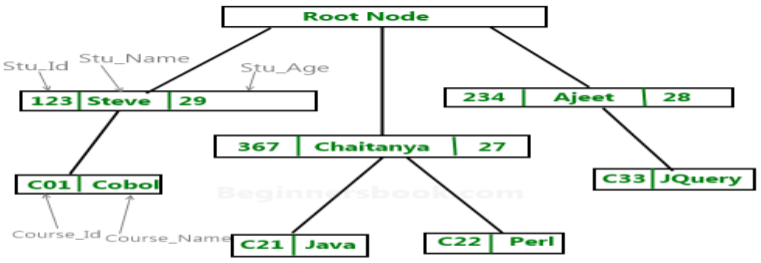
Here Stu_Id, Stu_Name & Stu_Age are attributes of table Student and Stu_Id, Course_Id & Course_Name are attributes of table Course. The rows with values are the records (commonly known as tuples).

Hierarchical model in DBMS

- In hierarchical model, data is organized into a tree like structure with each record is having one parent record and many children.
- The main drawback of this model is that, it can have only one to many relationships between nodes.

Sample Hierarchical Model Diagram:

Let us say we have few students and few courses and a course can be assigned to a single student only, however a student take any number of courses so this relationship becomes one to many.



Example of hierarchical data represented as relational tables: The above hierarchical model can be represented as relational tables like this:

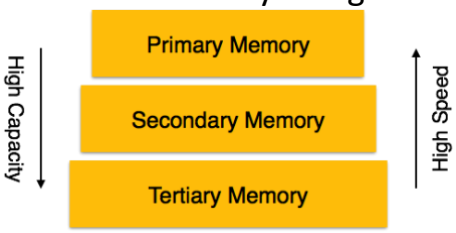
Stu_ID	Stu_Name	Stu_Age
123	Steve	29
367	Chaitanya	27
234	Ajeet	28

Course Table:

Course_Id	Course_Name	Stu_Id
C01	Cobol	123
C21	Java	367
C22	Perl	367
C23	JQuery	234

File Organization

- Databases are stored in file formats, which contain records.
- At physical level, the actual data is stored in electromagnetic format on some device.
- These storage devices can be broadly categorized into three types −



Primary Storage

- The memory storage that is directly accessible to the CPU comes under this category.
- CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset.
- This storage is typically very small, ultra-fast, and volatile.
- Primary storage requires continuous power supply in order to maintain its state.
- In case of a power failure, all its data is lost.

Secondary Storage

- Secondary storage devices are used to store data for future use or as backup.
- Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.

Tertiary Storage

- Tertiary storage is used to store huge volumes of data.
- Since such storage devices are external to the computer system, they are the slowest in speed.
- These storage devices are mostly used to take the back up of an entire system.
- Optical disks and magnetic tapes are widely used as tertiary storage.

Memory Hierarchy

- A computer system has a well-defined hierarchy of memory.
- A CPU has direct access to its main memory as well as its inbuilt registers.
- The access time of the main memory is obviously less than the CPU speed.
- To minimize this speed mismatch, cache memory is introduced.
- Cache memory provides the fastest access time and it contains data that is most frequently accessed by the CPU.
- The memory with the fastest access is the costliest one.
- Larger storage devices offer slow speed and they are less expensive, however they can store huge volumes of data as compared to CPU registers or cache memory.

Magnetic Disks

- Hard disk drives are the most common secondary storage devices in present computer systems.

- These are called magnetic disks because they use the concept of magnetization to store information.
- Hard disks consist of metal disks coated with magnetisable material.
- These disks are placed vertically on a spindle.
- A read/write head moves in between the disks and is used to magnetize or de-magnetize the spot under it.
- A magnetized spot can be recognized as 0 (zero) or 1 (one).

Hard Disks

- Hard disks are formatted in a well-defined order to store data efficiently.
- A hard disk plate has many concentric circles on it, called tracks. Every track is further divided into sectors.
- A sector on a hard disk typically stores 512 bytes of data.

Redundant Array of Independent Disks (RAID)

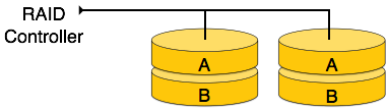
- RAID or Redundant Array of Independent Disks, is a technology to connect multiple secondary storage devices and use them as a single storage media.
- RAID consists of an array of disks in which multiple disks are connected together to achieve different goals.
- RAID levels define the use of disk arrays.

RAID 0

- In this level, a striped array of disks is implemented.
- The data is broken down into blocks and the blocks are distributed among disks.
- Each disk receives a block of data to write/read in parallel. It enhances the speed and performance of the storage device.
- There is no parity and backup in Level 0.

RAID 1

- RAID 1 uses mirroring techniques.
- When data is sent to a RAID controller, it sends a copy of data to all the disks in the array.
- RAID level 1 is also called mirroring and provides 100% redundancy in case of a failure.

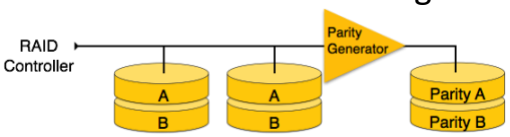


RAID 2

- RAID 2 records Error Correction Code using Hamming distance for its data, striped on different disks.
- Like level 0, each data bit in a word is recorded on a separate disk and ECC codes of the data words are stored on a different set of disks.
- Due to its complex structure and high cost, RAID 2 is not commercially available.

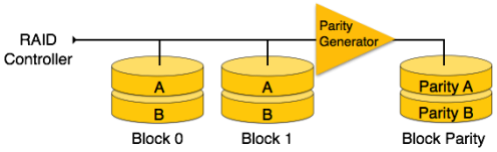
RAID 3

- RAID 3 stripes the data onto multiple disks.
- The parity bit generated for data word is stored on a different disk.
- This technique makes it to overcome single disk failures.



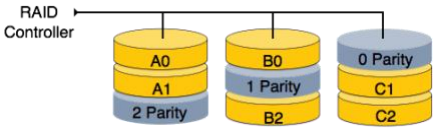
RAID 4

- In this level, an entire block of data is written onto data disks and then the parity is generated and stored on a different disk.
- Note that level 3 uses byte-level striping, whereas level 4 uses block-level striping.
- Both level 3 and level 4 require at least three disks to implement RAID.



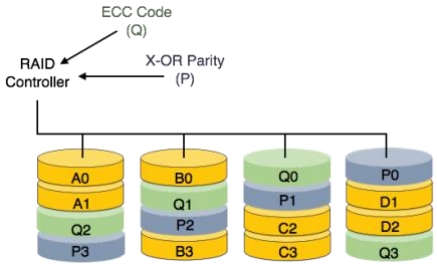
RAID 5

- RAID 5 writes whole data blocks onto different disks, but the parity bits generated for data block stripe are distributed among all the data disks rather than storing them on a different dedicated disk.



RAID 6

- RAID 6 is an extension of level 5.
- In this level, two independent parities are generated and stored in distributed fashion among multiple disks.
- Two parities provide additional fault tolerance.
- This level requires at least four disk drives to implement RAID.



File Structure

- Relative data and information is stored collectively in file formats.
- A file is a sequence of records stored in binary format.
- A disk drive is formatted into several blocks that can store records.
- File records are mapped onto those disk blocks.

File Organization

- File Organization defines how file records are mapped onto disk blocks.
- We have four types of File Organization to organize file records –
    a. Heap File Organization
    b. Sequential File System
    c. Hash File System
    d. Clustered File System

Heap File Organization

- When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details.
- File records can be placed anywhere in that memory area.
- It is the responsibility of the software to manage the records.
- Heap File does not support any ordering, sequencing, or indexing on its own.

Sequential File Organization

- Every file record contains a data field (attribute) to uniquely identify that record.
- In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key.
- Practically, it is not possible to store all the records sequentially in physical form.

Hash File Organization

- Hash File Organization uses Hash function computation on some fields of the records.

- The output of the hash function determines the location of disk block where the records are to be placed.

Clustered File Organization

- Clustered file organization is not considered good for large databases.
- In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

File Operations

- Operations on database files can be broadly classified into two categories –
    - Update Operations
    - Retrieval Operations
- Update operations change the data values by insertion, deletion, or update.
- Retrieval operations, on the other hand, do not alter the data but retrieve them after optional conditional filtering.
- In both types of operations, selection plays a significant role.
- Other than creation and deletion of a file, there could be several operations, which can be done on files.
    - Open − A file can be opened in one of the two modes, read mode or write mode. In read mode, the operating system does not allow anyone to alter data. In other words, data is read only. Files opened in read mode can be shared among several entities. Write mode allows data modification. Files opened in write mode can be read but cannot be shared.
    - Locate − Every file has a file pointer, which tells the current position where the data is to be read or written. This pointer can be adjusted accordingly. Using find (seek) operation, it can be moved forward or backward.
    - Read − By default, when files are opened in read mode, the file pointer points to the beginning of the file. There are options where the user can tell the operating system where to locate the file pointer at the time of opening a file. The very next data to the file pointer is read.
    - Write − User can select to open a file in write mode, which enables them to edit its contents. It can be deletion, insertion, or modification. The file pointer can be located at the time of opening or can be dynamically changed if the operating system allows to do so.

- Close − This is the most important operation from the operating system's point of view. When a request to close a file is generated, the operating system
    - removes all the locks (if in shared mode),
    - saves the data (if altered) to the secondary storage media, and
    - releases all the buffers and file handlers associated with the file.
- The organization of data inside a file plays a major role here.
- The process to locate the file pointer to a desired record inside a file various based on whether the records are arranged sequentially or clustered.

Indexing

- We know that data is stored in the form of records.
- Every record has a key field, which helps it to be recognized uniquely.
- Indexing is a data structure technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done.
- Indexing in database systems is similar to what we see in books.
- Indexing is defined based on its indexing attributes. Indexing can be of the following types −
    - Primary Index − Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.
    - Secondary Index − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
    - Clustering Index − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.
- Ordered Indexing is of two types −
    - Dense Index
    - Sparse Index

Dense Index

- In dense index, there is an index record for every search key value in the database.
- This makes searching faster but requires more space to store index records itself.

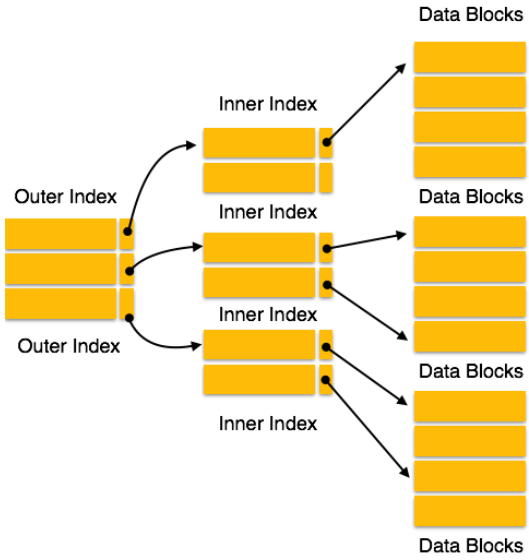
- Index records contain search key value and a pointer to the actual record on the disk.

Sparse Index

- In sparse index, index records are not created for every search key.
- An index record here contains a search key and an actual pointer to the data on the disk.
- To search a record, we first proceed by index record and reach at the actual location of the data.
- If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.

Multilevel Index

- Index records comprise search-key values and data pointers.
- Multilevel index is stored on the disk along with the actual database files.
- As the size of the database grows, so does the size of the indices.
- There is an immense need to keep the index records in the main memory so as to speed up the search operations.
- If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



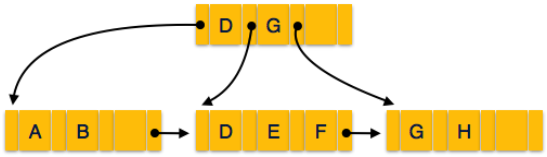
- Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

B^+ Tree

- A B^+ tree is a balanced binary search tree that follows a multi-level index format.
- The leaf nodes of a B^+ tree denote actual data pointers.
- B^+ tree ensures that all leaf nodes remain at the same height, thus balanced.
- Additionally, the leaf nodes are linked using a link list; therefore, a B^+ tree can support random access as well as sequential access.

Structure of B^+ Tree

- Every leaf node is at equal distance from the root node.
- A B^+ tree is of the order n where n is fixed for every B^+ tree.



Internal nodes

- Internal (non-leaf) nodes contain at least ⌈n/2⌉ pointers, except the root node.
- At most, an internal node can contain n pointers.

Leaf nodes

- Leaf nodes contain at least ⌈n/2⌉ record pointers and ⌈n/2⌉ key values.
- At most, a leaf node can contain n record pointers and n key values.

- Every leaf node contains one block pointer P to point to next leaf node and forms a linked list.

B^+ Tree Insertion

- B^+ trees are filled from bottom and each entry is done at the leaf node.
- If a leaf node overflows −
  - Split node into two parts.
  - Partition at i = $\lfloor (m+1)/2 \rfloor$.
  - First i entries are stored in one node.
  - Rest of the entries (i+1 onwards) are moved to a new node.
  - i^{th} key is duplicated at the parent of the leaf.
- If a non-leaf node overflows −
  - Split node into two parts.
  - Partition the node at i = $\lceil (m+1)/2 \rceil$.
  - Entries up to i are kept in one node.
  - Rest of the entries are moved to a new node.

B^+ Tree Deletion

- B^+ tree entries are deleted at the leaf nodes.
- The target entry is searched and deleted.
  - If it is an internal node, delete and replace with the entry from the left position.
- After deletion, underflow is tested,
  - If underflow occurs, distribute the entries from the nodes left to it.
- If distribution is not possible from left, then
  - Distribute from the nodes right to it.
- If distribution is not possible from left or from right, then
  - Merge the node with left and right to it.

REFERENCES

1. Silberschatz A., Korth H. and Sudarshan S., "Database System Concepts", Tata McGraw Hill, 2011.
2. Elmasri R. and Navathe S.B., "Fundamentals of Database Systems", Pearson Education, 2016.
3. www.w3schools.com
4. www.guru99.com
5. www.tutorialspoint.com
6. www.beginnersbook.com
7. www.ecomputernotes.com

----------**********----------