CSA05 - DATABASE MANAGEMENT SYSTEMS

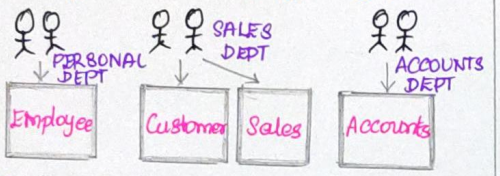
TOPICS	CONCEPT MAP NO.
INTRODUCTION TO FILE AND DATABASE File Systems - Definition, Figure, Disadvantage Database Systems - Definition, Figure - Characteristics, Advantages, Components, - History of DBMS, View of Data, Application File System Vs DBMS (table)	1
DATABASE SYSTEM STRUCTURE Terminologies - Data, Database, DBMS - structure & Diagram SOFTWARE ARCHITECTURE OF A TYPICAL DBMS - Two tier, three-tier Architecture	2
DATA MODELS - Hierarchical, Relational, Network, Object, ER Model EER DIAGRAMS - Subclass, Superclass, Specialization, Generalization, Aggregation SCHEMAS AND INSTANCES	3
ER MODEL - Entity, Attributes, Relationships	4
RELATIONSHIPS INTRODUCTION TO NETWORK & HIERARCHICAL MODELS - Hierarchical, Network Model, Figure - Differences - Notations & General ER Diagram	5
RELATIONAL MODEL INTRODUCTION TO RELATIONAL MODEL - Domain, Attribute, Instance, Schema Key.	6

TOPICS	CONCEPT MAP NO.
RELATIONAL ALGEBRA - Relational operators, JOINS	6
RELATIONAL CALCULUS - Tuple & Domain Relational Calculus SQL - DATA DEFINITION - DDL, DML, DCL, DQL, TCL QUERIES IN SQL - UPDATES - VIEWS - SUBQUERIES - CONSTRAINTS - INTEGRITY AND SECURITY	7
KEYS : TYPES - Primary, Foreign - Eg. of DDL, DML, Join, 4NF	8
RELATIONAL DATABASE DESIGN FUNCTIONAL DEPENDENCIES NORMALIZATION FOR RELATIONAL DATABASE - Normalization, 1NF, 2NF, 3NF, BCNF	9
MULTI-VALUED DEPENDENCY _ JOIN DEPENDENCIES - DEFINITION OF 5NF	10
RECORD STORAGE - Record, types - Fixed & Variable length records - Memory Hierarchy (Primary, types)	11
PRIMARY FILE ORGANIZATION - Types : Sequential, Heap, Hash SECONDARY STORAGE DEVICES - HDD, SSD, Magnetic Tapes, Access OPERATIONS ON FILES : Open, Close - Retrieval, Update HEAP FILE - structure & Diagram SORTED FILES - Sort (Quick, Merge)	12
HASHING TECHNIQUES - Terminologies, - Static, Bucket Overflow & Collision, - Types & Dynamic Hashing	13

TOPICS	CONCEPT MAP NO.
INDEX STRUCTURE FOR FILES : Index, Types DIFFERENT TYPES OF INDEXES - Single level : Primary, Secondary - Dynamic Multilevel	14
SEQUENCES & INDEX : Syntax, Eg.	
B-TREE : Eg, Operations - Insert, Delete B+ TREE : Eg, Operations - Insert, Delete	15
QUERY PROCESSING : Steps, Measures - Selection, Evaluation of Expressions QUERY OPTIMIZATION ALGORITHMS Transformation, Estimating statistics	16
HEURISTICS & COST ESTIMATES IN QUERY OPTIMIZATION - Heuristic Rules, Steps, Cost estimates, cost components IMPLEMENTATION OF DATABASE CONNECTIVITY - Database connectivity, JDBC CREATE, DELETE, UPDATE	17
TRANSACTION MANAGEMENT - T. PROCESSING - INTRODUCTION - Definition & Eg. NEED FOR CONCURRENCY CONTROL - Concurrency Control, States DESIRABLE PROPERTIES OF TRANSACTION RAID - Types	18
SCHEDULE AND RECOVERABILITY - Definition, Types : Serial, Non-Serial SERIALIZABLE AND SCHEDULES - Conflict, View Serializability	19
CONCURRENCY CONTROL - TYPES OF LOCKS - Shared, Exclusive, Binary lock TWO PHASE LOCKING - Growing, Shrinking	20
DEADLOCK - Definition, Detection, Prevention TIME STAMP-BASED CONCURRENCY CONTROL: Eg RECOVERY TECHNIQUES - Commit, Savepoint IMMEDIATE, DEFERRED UPDATE - Definition SHADOW PAGING - Definition Difference	21
INNOVATIVE TOPICS - MongoDB, Amazon RDS	22

TOPIC : INTRODUCTION TO FILE AND DATABASE

FILE SYSTEMS
- arrange files - storage medium
- organize data, types of files



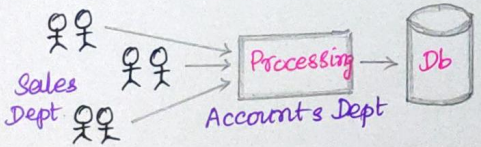
DISADVANTAGES
- Data redundancy & inconsistency
- Difficulty in accessing data
- Data Isolation
- Integrity Problems
- Atomicity of updates
- Concurrent access of data
- Security problems

DATABASE SYSTEMS
DATABASE MANAGEMENT SYSTEMS
- Software - manage database
- eg: MySQL, Oracle, IBM DB2

DATA : raw fact

INFORMATION: processed data

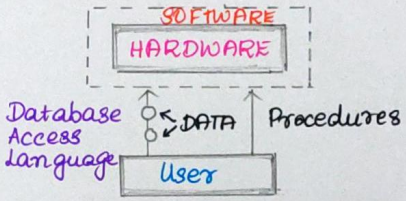
DATABASE: collection of interrelated data



CHARACTERISTICS OF DBMS
- Centralized control of data
- shared among users
- data redundancy : duplication
- data persistency
- supports multiple data types

ADVANTAGES OF DBMS
- Easy retrieval
- minimum redundancy
- Data Integrity
- Data Security
- Reduced Development Time
- Concurrent access
- Data consistency

COMPONENTS OF DBMS



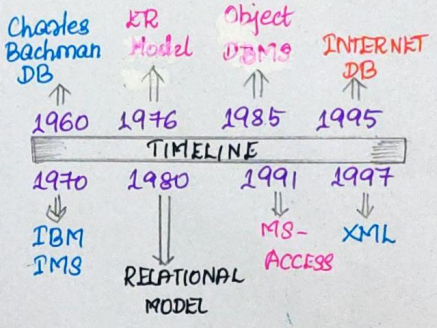
- five major components

1. **HARDWARE**
   - physical component
   - computer, hard disks
2. **SOFTWARE**
   - interface to store, access
3. **DATA**
   - resource
   - DBMS store and utilize data
   - metadata - data about data
4. **PROCEDURES**
   - instructions to use DBMS
   - setup and install DBMS

5. DATABASE ACCESS LANGUAGE
- language to write commands ⇓ to access, insert data in any database

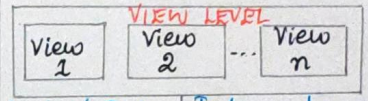
Users
- Database Administrators
- Application Programmer
- End User

HISTORY OF DATABASE SYSTEMS



APPLICATIONS OF DBMS
- Railway Reservation System
- Banking, Education
- Credit Card Exchanges
- Social Media sites
- Online Shopping
- Human Resource Management

VIEW OF DATA: Data Abstraction



Logical Data | Independence

LOGICAL LEVEL

Physical Data | Independence

PHYSICAL LEVEL

PHYSICAL LEVEL - how data stored, complex DS

LOGICAL LEVEL - what data is stored, relationship

VIEW LEVEL - part of the database

FILE SYSTEMS	DBMS
- software: manages & organizes files.	- software for managing database.
- Redundant data can be present.	- There is no redundant data.
- It doesn't provide backup & recovery of data if it is lost.	- It provides backup & recovery of data if it is lost.
- There is no efficient query processing.	- Efficient query processing is present
- less Data consistency	- More data consistency due to normalization
- Only one user can access	- Multiple users can access

DATABASE SYSTEM STRUCTURE.

DATA: RAW Data | information

DATABASE: Collection of interrelated data

→ used to insert, retrive & delete data.

→ organize the data in the form of a table, schema views & reports etc

Database Management system.

→ Software used to Manage the database.

Ex:- MySQL
Oracle etc

→ Provides interface

→ database creation, storing, updating, protection & security.

DBMS ARCHITECTURE

→ Database system runs on Computer system.

- Database system.
  - centralized
  - client / server
  - parallel Computer Architecture.

DB:- Defining, storing, Manupulating & protecting information.

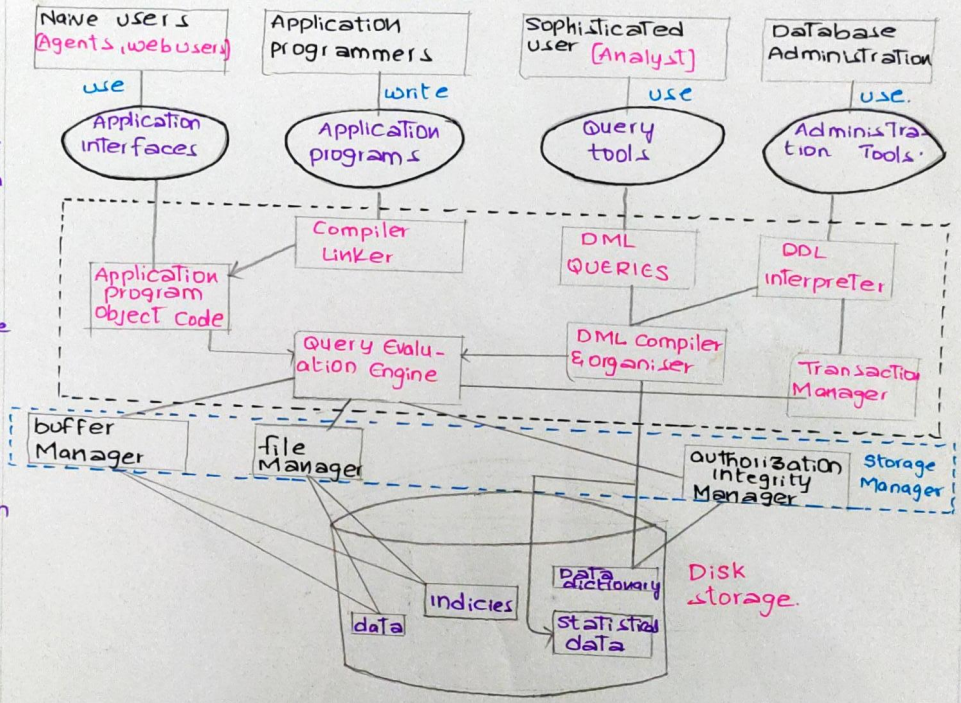
→ Three Components.

PHYSICAL SYSTEM IMPLEMENTATION.

→ Data file store

→ Data dictionary - Meta data - structure - schema - database.

→ Indicies - fast access - values.



1. **Query processor**: interprets the queries into instructions.
   - DML Compiler: processes DML statement into instructions
   - DDL interpreter: processes DDL statement into a set of table [Meta data]
   - Query optimization: low cost evaluation plan
   - Query Evaluation Engine: Executes low level instruction.

2. **Storage Manager**.
   - → Interface b/w data stored & Queries received.
   - → Consistency & integrity
   - → update, store, delete & retrieve.
   - Authorization Manager: ensures role-based access control
   - Integrity Manager: checks the integrity constraints
   - Transaction Manager: Controls Concurrent access
   - File Manager: Manages space & data structure.
   - Buffer Manager: cache Memory transfer data b/w secondary Memory.

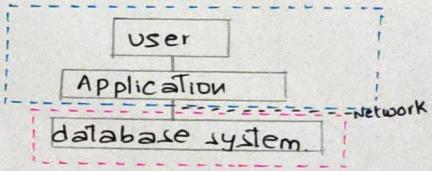
3. **Disk storage.**
   - Datafile: stores data.
   - Data dictionary: structure information - repository - Metada ta.
   - Indicies: faster retrieval of data.

Software Architecture of a Typical DBMS

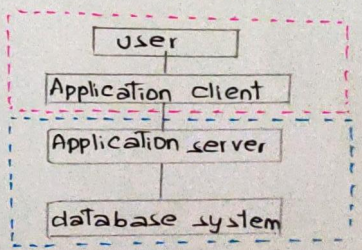
DB Applications → Two parts / Three parts

1. **Two-tier Architecture.**
   - → Application - client Machine
   - → Database - server Machine
   - → Query language statement

   Ex:- ODBC, JDBC.



2. **Three Tier Architecture.**
   - → Frontend - client Machine
   - → Communicates - Application server
   - → Database system - server Machine



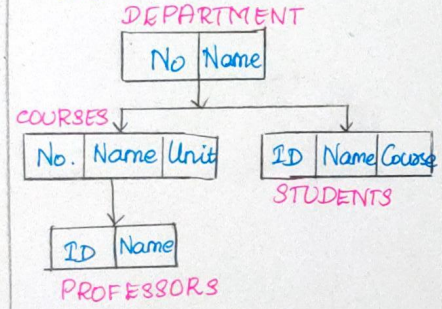
DATA MODELS

* collection of concepts that can be used to describe structure of a database.

TYPES:

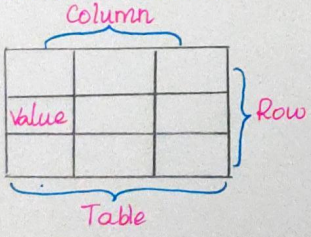
① HIERARCHICAL MODEL

* data - hierarchical tree structure



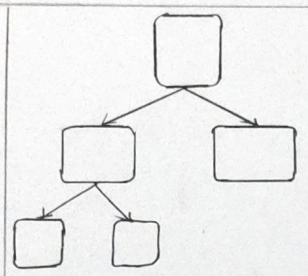
② RELATIONAL MODEL

* data - organized in 2-dimensional tables - called relations.



③ NETWORK MODEL

* allows a record to have more than one parent



④ OBJECT MODEL

* stores the data - form of objects, classes, inheritance.
* used in File Management System.

⑤ ER Model - EER.

* objects
* relationships.

DATA MODEL - SCHEMA & INSTANCE.

INSTANCE: data stored in db at particular time.

SCHEMA: Overall design of a database.

* logical view.
* contains - table, foreign key, primary key, views, columns, data types.

STUDENT:

Name	Student-Number	class	Major

COURSE

Name	Number	Credit-hours	Dept.

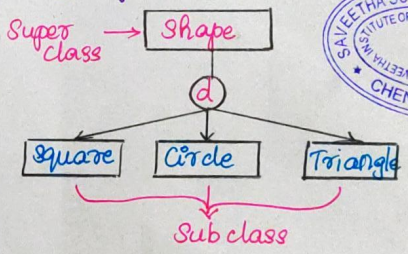
ENHANCED ENTITY RELATIONSHIP MODEL (EER)

* incorporates extensions to ER model.
* represents following concepts

① Subclass & Superclass

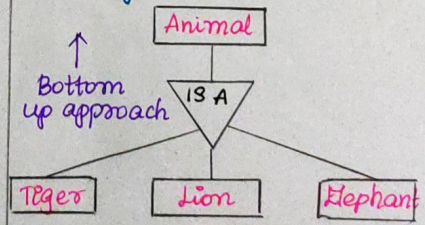
- concept of inheritance.

ⓐ symbol.



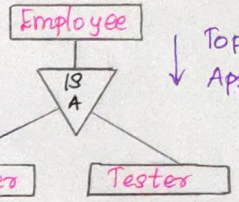
② Generalization

- bottom approach
- process of generalizing entities - contains properties of all generalized entities.



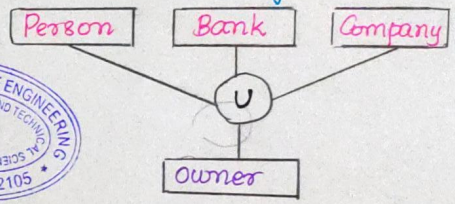
Specialization

- top down approach



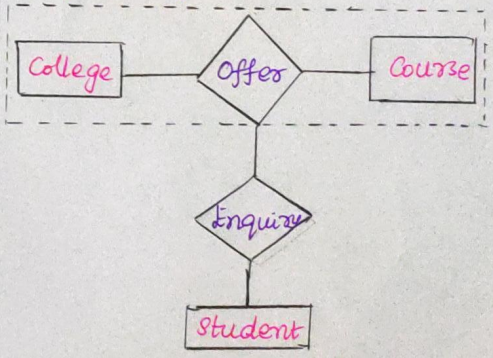
③ Category or Union

* represents single super class/ sub class relationship with more than one super class.



④ Aggregation

* represent a relationship between a whole object & its component parts.



- Relation between College & Course - Entity in relation with Student

ER Model

* Entity-Relationship model
* used to define data elements & relationship for a specified system.
* Devlops conceptual design for the database.

eg: school database

Entity: student

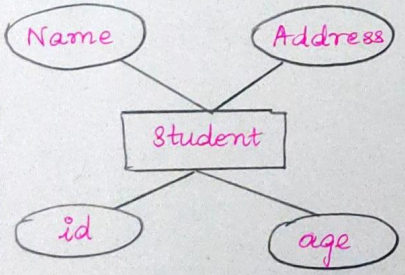
attributes

address, name, id, age

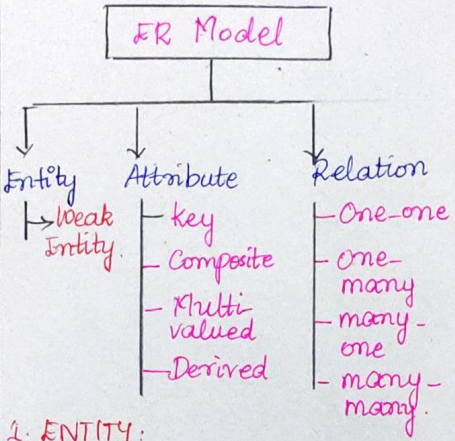
Entity: address

attributes

city, street name, pincode

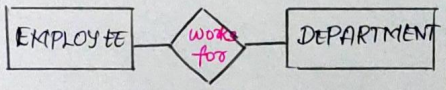


Components of ER diagram



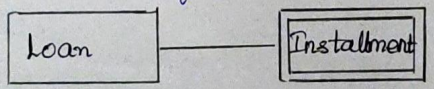
1. ENTITY:

* Object, class, person/place.
* rectangle shape.
* employee, department: eg



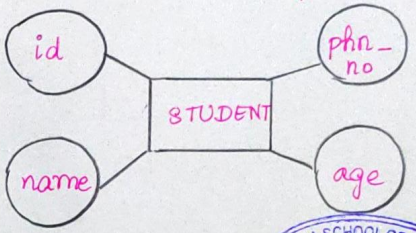
a) WEAK ENTITY

* Entity that depends on another entity.
* doesn't contain any key attribute of its own.



2. ATTRIBUTE

* describes property of an entity.
* eclipse shape.
* eg: id, age, Contact number, name → attributes of student.

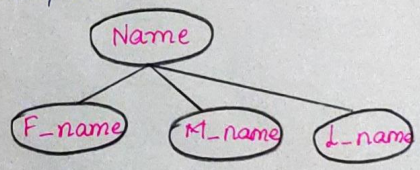


a) KEY ATTRIBUTE

* represents main charactoristics of an entity.
* primary key.
* ellipse with text underlined
* eg: id.

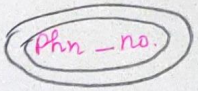
b) COMPOSITE ATTRIBUTE

* composed of many other attributes.
* ellipse.



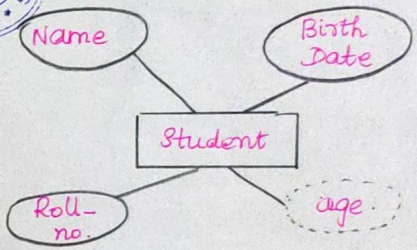
c) MULTIVALUED ATTRIBUTE

* attribute has more than one value.
* double oval.
* eg: student - more phone number.



d) DERIVED ATTRIBUTE

* Attribute derived from other attribute.
* dashed ellipse.
* eg: Person's age is derived from Date of birth.



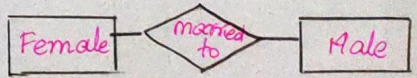
3. RELATIONSHIP

* describes relation between entities.

TYPES

* ONE-TO-ONE

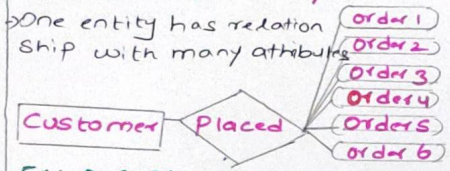
⇒ One instance of an entity is associated with the relationship.



RELATIONSHIP TYPES

* ONE - TO - MANY

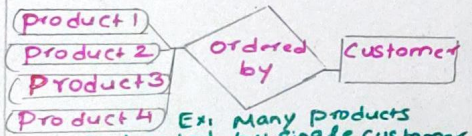
One entity has relationship with many attributes



Ex: A customer may place many orders

* MANY - TO - ONE

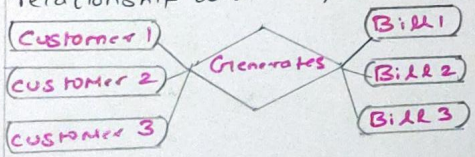
→ many attributes has relationship with one attributes



Ex: Many products ordered by single customer

* MANY - TO - MANY

→ many attributes has a relationship with many attributes



Ex: Many customer may generate many bills

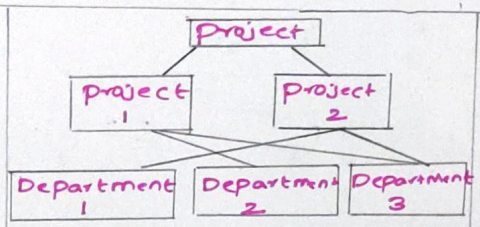
INTRODUCTION TO NETWORK AND HIERACHICAL MODELS

NETWORK DATA MODEL

→ Represents complex data relationship

→ Handles many to many relationship

→ Easy to Search and Navigate



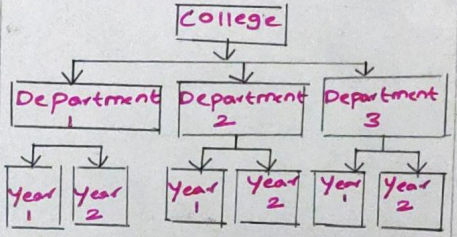
Ex: A project May have relationship with various departments

HIERACHICAL DATA MODEL

→ Data organised like tree Structure

→ Handles One to many relationship

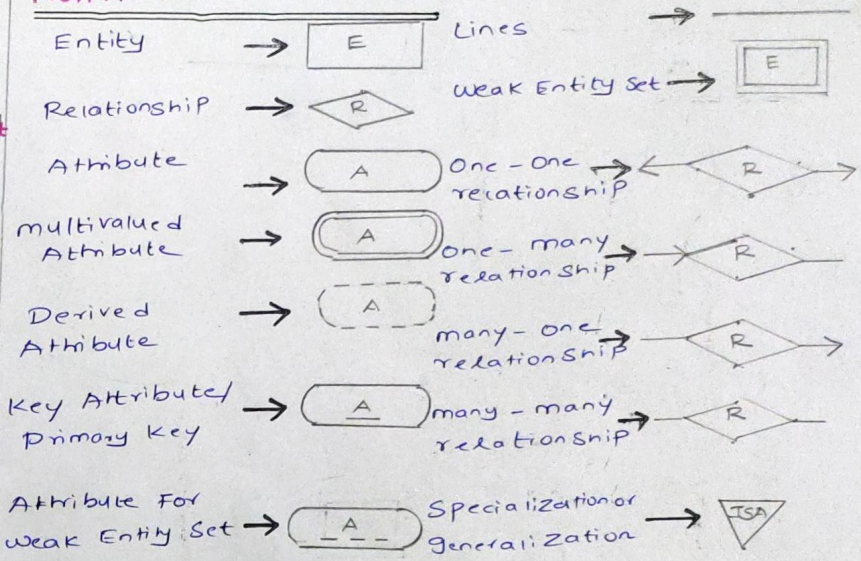
→ Searching and navigation is not easy and complex



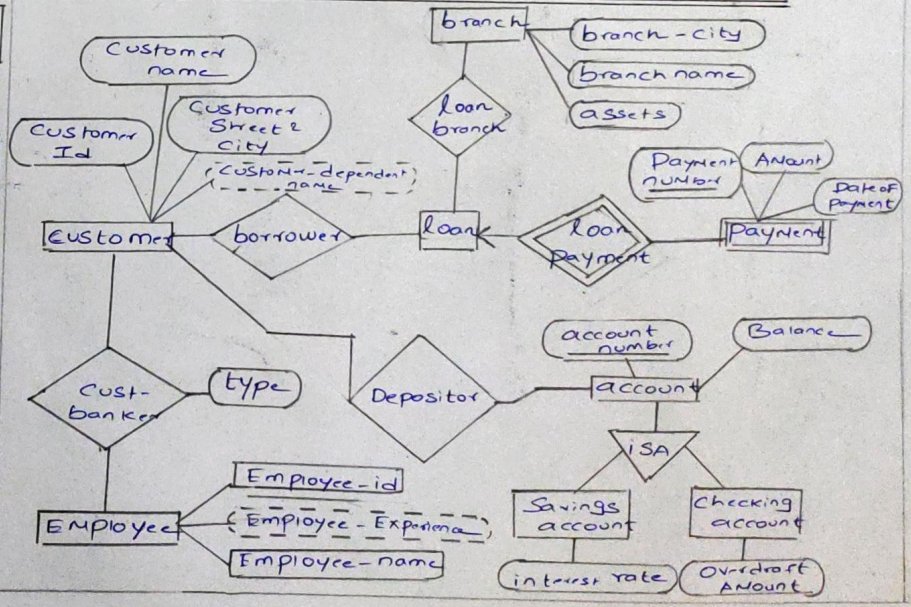
Ex: Each department in college may have various year Students

NETWORK DATA MODEL	HIERACHICAL DATA MODEL
① Data allows linking Multiple records - graph Structure	Data Organised like tree Structure
② Many to One and one to many, and many to many supported	only One to many relationship is supported
③ Data access is easier	Data access is complex
④ Flexible	Less Flexible

NOTATIONS OF ER DIAGRAM



EXAMPLE FOR ER DIAGRAM - BANKING



UNIT – 2 RELATIONAL MODEL

1. INTRODUCTION

* Relational model can represent as a table with columns and rows.
* Each row – tuple.
* Each table – name / attribute.

TERMINOLOGIES

* DOMAIN: set of atomic values that an attribute can take.
* ATTRIBUTE: contains name of column in a particular table.
* RELATIONAL INSTANCE:
  * represented by a finite set of tuples.
* Relational schema:
  Name of relation + name of all columns / attributes.
* Relational key:
  identify the row in the relation uniquely.

Eg: STUDENT Relation

Name	ROLL_NO	PH_NO	ADDRESS	AGE
Ram	1423	1321689	Noida	24
Mohan	1679	6971321	Delhi	36
Reeta	5234	1783261	Chennai	48

Attributes: NAME, ROLL_NO, PH_NO, ADDRESS, AGE

Instance of schema STUDENT

⇓

5 tuples.

RELATIONAL ALGEBRA

* procedural query language.
* operators are used to perform queries.

OPERATORS:

① Selection (σ)

* used to select the required tuple / row from the table.

$\sigma_{condition}$ (Relation / Table Name)

② Project Operator (Π)

* used to retrieve data from a column of a table.

$\Pi_{col.name1 \ldots col.nameN}$ (table_name)

③ Union Operation (∪)

* used to select all rows (tuples) from two tables (relations)

table_1 ∪ table_2

④ Intersection Operator (∩)

* used to select common rows (tuples) from two tables (relations)

table_1 ∩ table_2

⑤ Set Difference Operator (−)

* Suppose 2 tuples R & S.
* ∩ contains all tuples that are in R but not in S.

R − S

table_1 − table_2.

⑥ Cartesian product (×)

* used to combine each row in one table with each row in other table
* cross product

R1 × R2

⑦ Rename (ρ)

* used to rename a relation or an attribute of a relation.

ρ(new_relation, old_relation)

JOIN OPERATION (⋈)

cartesian product followed by a selection criterion.

JOIN OPERATION

- Natural Join
- Outer Join
  - Left outer
  - Right outer
  - Full outer
- Equi Join

Natural join: performed if there is a common attribute between the relations.

Left Outer Join (⟕)

* allows keeping all tuple in left relation.

Right Outer Join (A ⟖ B)

* operations allows keeping all tuple in right relation.

Full outer join (⟗)

all tuples from both relations are included in the result, irrespective of the matching condition.

Equi Join (=)

* inner join
* based on matched data as per equality condition.

RELATIONAL CALCULUS

* non-procedural query language.

Types

1. Tuple Relational Calculus.

* used for selecting those tuples that satisfy the given condition.

Syntax: {T | Condition}

Eg: T.name | Student(T) AND T.age > 17

* fetch names of students.

2. Domain Relational Calculus

* records are filtered based on the domains of attributes & not based on tuple values.

Syntax: {c1, c2, ... cn | F(c1, ..., cn)}

c1, c2 : domain of attributes

F : formula.

eg: {<name, age> | ∈ Student ∧ age > 17}

SQL

* Structured Query Language
* storing & managing data in RDBMS.

DATA TYPES

* CHAR(n); VARCHAR(n); FLOAT

TYPES :

① DATA DEFINITION LANGUAGE (DDL) :

(i) CREATE

create new table in database

```
create table table_name
(col_name datatypes[,...]);
```

(ii) DROP : delete record

```
DROP TABLE table_name
```

(iii) ALTER : alter db structure

```
ALTER TABLE table_name
ADD col_name COL_definition;
```

(iv) TRUNCATE : delete all rows

```
TRUNCATE TABLE table_name.
```

② DATA MANIPULATION LANGUAGE

* used to modify database

(i) INSERT:

```
INSERT INTO TABLE_NAME
VALUES (val1, val2, ... val.N);
```

(ii) UPDATE:

```
UPDATE table_name SET
[col_name 1 = val.1, .... col_name
= val N][WHERE CONDITION]
```

(iii) DELETE:

```
DELETE FROM table_name
[WHERE condition];
```

③ DATA CONTROL LANGUAGE (DCL)

(i) GRANT : give user access privileges to a database

(ii) REVOKE : used to take back permissions from the user.

④ TRANSACTION CONTROL LANGUAGE (TCL)

(i) COMMIT : used to save all transactions to database

(ii) ROLLBACK : undo transactions that have not already been saved

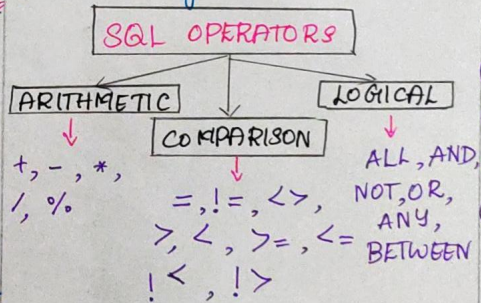
(iii) SAVEPOINT : roll back transaction back to a certain point

⑤ DATA QUERY LANGUAGE (DQL)

fetch data from database

(i) SELECT

select attribute based on condition by WHERE clause.



SQL UPDATE: modify data is already in the database.

```
UPDATE table_name
SET col1 = value1, col2 = value2
WHERE condition;
```

VIEWS IN SQL

* Virtual table.
* contains rows & columns

(i) CREATE VIEW

(ii) DELETE VIEW ⇒ DROP VIEW

SUB-QUERY

* inner query / nested query is a query within another SQL query & embedded within WHERE
* used with SELECT, INSERT, UPDATE, DELETE & operators (=, >, <)

CONSTRAINTS

* used to specify rules for data in a table.

(1) NOT NULL: value cannot be empty

(2) UNIQUE: duplicate values not allowed.

(3) PRIMARY KEY: Not null + Unique.

(4) FOREIGN KEY: referential integrity

(5) CHECK: condition checked.

(6) DEFAULT: default value inserted

(7) CREATE INDEX: create index

INTEGRITY AND SECURITY

* correctness, consistency.

(i) Entity integrity, (ii) Referential integrity, (iii) Domain integrity.

* protection from malicious attempts to steal / modify data.

4NF

- removes non-trivial multivalued dependency.
- table must be in BCNF & no non-trivial MVD.

RegNo	PhoneNo	Qualification
1	P1	Diploma
1	P1	B.Tech
1	P1	M.Tech
1	P2	Diploma
1	P2	B.Tech
1	P2	M.Tech

Above table properties,
- in BCNF form
- no functional dependency
- But, has non-trivial MVD.

RegNo → PhoneNo

RegNo → Qualification

4NF decomposition

Table 1

RegNo	PhoneNo
1	P1
1	P2

Table 2

RegNo	Qualification
1	Diploma
1	B.Tech
1	M.Tech

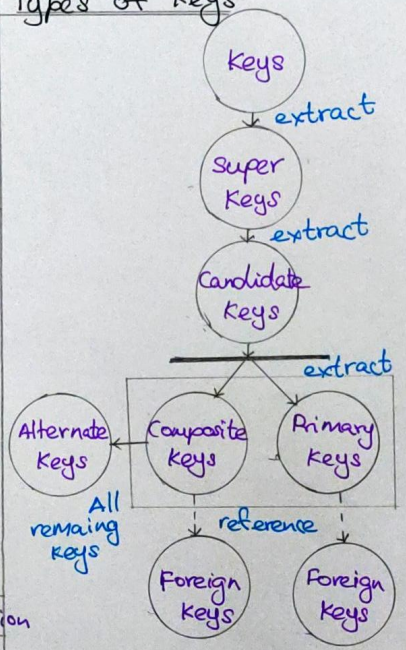
Table 1 : RegNo ∪ PhoneNo

Table 2: RegNo ∪ Qualification.

Keys in DBMS

- item used to uniquely identifies a record.
- used to fetch records from table according to conditions.

Types of keys



Super keys

- used to identify an attribute.

Candidate keys

- all attributes selected as primary key.

Primary key

- values must be unique.
- values cannot be NULL.
- given when a new record inserted.

Alternate key

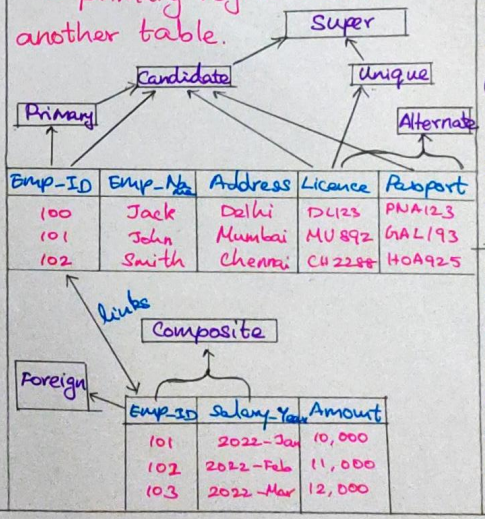
* Candidate key − Primary key = alternate key
* primary keys but not selected as primary keys.

Composite key

- set of two or more attributes that are together.

Foreign key

- primary key linked with another table.



Data Definition Language (DDL)

- Create, Drop, Alter, Truncate

Create - used to create table / database

```
CREATE TABLE employee;
CREATE DATABASE company;
```

Drop - delete structure & records.

```
DROP TABLE employee;
```

Alter - used to change structure of table.

```
ALTER TABLE student ADD (address varchar(20));
```

Truncate - delete all rows

```
TRUNCATE TABLE employee;
```

Data Manipulation Language (DML)

- Insert, Update, Delete

```
INSERT INTO employee (Name, Design) VALUES ("John", "SE");
UPDATE employee SET Name= "John" WHERE ID = '101';
DELETE FROM employee WHERE Name = "John";
```

Joins

Inner : select ename, loc from emp join dept on emp.deptno = dept.deptno where loc = "chen";

Outer : Select rowno, ename, loc, job, mgr from emp FULL OUTER JOIN dept on emp.deptno = dept.deptno;

NORMAL FORMS IN DBMS

RELATIONAL DATABASE DESIGN

Normalization

* process of minimizing redundancy from a relation or set of relations.
* Redundancy cause
  - ↳ insertion
  - → deletion
  - → update anomalies
* Normal forms
  - eliminate / reduce redundancy in database tables.

1. FIRST NORMAL FORM

* A relation is in first normal form if every attribute in that relation is singled valued attribute.
* If a relation contains multi-valued attribute, it violates first normal form.

Table 1: Relation STUDENT
Multi-valued attributed STUD_PHONE

STUD_NO	STUD_NAME	STUD_PHONE	STUD_COUNTRY
823	RAM	971621023 9843050634	INDIA INDIA
824	SURESH	9888456211	INDIA

↓ Conversion to first NORMAL FORM

STUD_NO	STUD_NAME	STUD_PHONE	STUD_COUNTRY
823	RAM	971621023	INDIA
823	RAM	9843050634	INDIA
824	SURESH	9888456211	INDIA

2. SECOND NORMAL FORM

* A relation must be in 1NF.
* Relation must not contain any partial dependency.
* no partial dependency

eg: Table: 2

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

↓

Many course with same course fees.
Split Table as,
Table 1: STUD_NO, COURSE_NO
Table 2: COURSE_NO, COURSE_NO

TABLE 1	
STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1

TABLE 2	
COORSE_NO	COORSE_FEE
C1	1000
C2	1500
C3	1000
C4	2000
C5	2000

3. THIRD NORMAL FORM

A relation is in 3NF,
* If there is no transitive dependency for non-prime attributes.
* A relation must be in 2NF
* functional dependency $X \to Y$
  - X is a superkey.
  - Y is a prime attribute.

Eg: Table 3.

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Transitive Dependency - If $A \to B$ & $B \to C$ are two FDs, $A \to C$ is TD.

Eg: 1. Relation STUDENT
FD set : {STUD_NO → STUD_NAME, STUD_NO → STUD_STATE, STUD_STATE → STUD_COUNTRY, STUD_NO → STUD_AGE}
candidate key : {STUD_NO}
↓ decompesed as
STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
STATE COUNTRY (STATE, COUNTRY)

4. BOYCE – CODD NORMAL FORM

* A relation in R is in BCNF
  - → if R is in 3NF.
  - → for every FD, LHS is a super key.
  - → iff in every non-trivial functional dependency $X \to Y$, X is a super key.

Eg: Find highest normal form of a relation R(A, B, C, D, E) with FD set as {BC→D, AC→BE, B→E}.
* AC : candidate key.
* prime attributes {A, C}
  non-prime : {B, D, E}
* 1NF : No multi-valued attribute
2NF : BC→D, AC→BE, B→E
Highest normal form → SECOND NORMAL FORM.

N. Deepa, AR & VR

TOPIC: FUNCTIONAL DEPENDENCES, JOIN DEPENDENCIES - DEFINITION OF 5NF, MULTI-VALUED DEPENDENCY

FUNCTIONAL DEPENDENCES

DEFINITION:

→ Constraint between 2 Set of attributes from database.

Relational database schema
- n attributes
- $A_1, A_2 \cdots A_n$

Universal schema
$R = \{A_1, A_2, A_3 \cdots A_n\}$

denoted by
$X \rightarrow Y$, between 2 sets of attributes.

X ⊆ Y – subset of R on tuples from state rel R

Constraint:

Tuples t_1 & t_2 in r
$t_1[X] = t_2[X]$ also
$t_1[Y] = t_2[Y]$

⇒ Functional dependency (FD or f.d.)

↳ Properties → Semantics
→ Relation States r(R)

Relation extension that satisfy fD constraints are called legal relation states (or) legal extension of R.

Attributes
{State, Driver_license-number} → SSn

SSn – held for adult in US

Used whenever particular attribute required.

Some attribute Changes

Ex: FD Zip_Code → Area_Code

used to exist as a relationship between postal code & telephone number in US.

→ telephone code may change.

Eg: SSn → Ename
Pnumber → {Pname, Plocation}
{SSn, Pnumber} → Hours

Above are functional dependencies

TEACH

Teacher	Course	Text
Smith	Data Structure	Bartram
Smith	Data management	Martin
Hall	Compilers	Hoffman
Brown	Data structures	Horowitz

Relation State of Teach with a possible functional dependency.

TEXT → COURSE

TEACHER → COURSE is ruled out.

A single Counter Example for functional dependency is shown in table.

JOIN DEPENDENCIES

The binary decompositions which follows BCNF will follow 4CNF as well.

↳ a Relation schema with R which violates BCNF and no nontrivial R there comes a join dependency.

↳ This also follows the multiway decomposition into fifth Normal form (5NF).

A Join dependency (JD)

JD $(R_1, R_2, \cdots R_n)$

Specified on

Relation schema R.

Specifies on

State r of R.

Should have nonadditive join decomposition into

$R_1, R_2, \cdots R_n$

Hence r have

$* (\pi_{R_1}(r), \pi_{R_2}(r), \cdots \pi_{R_n}(r)) = r$

Multivalued dependency (MVD)

JD where n = 2

JD (R_1, R_2)

(implies)

MVD $(R_1 \cap R_2) \twoheadrightarrow (R_1 - R_2)$

(or)

Symmetry, $(R_1 \cap R_2) \twoheadrightarrow (R_2 - R_1)$

5NF – A Definition.

Also called Project-join normal form (PJNF)

Set F of functional, multivalued and join dependency if for every nontrivial join dependency

JD $(R_1, R_2 \cdots R_n)$ in F^+

R_i is a superkey of R.

SUPPLY all-key relation where

supply is an example of supermarket that can hold part p to use relations.

RECORD STORAGE

- files of records stored on magnetic disk.
- Storage medium, form of storage hierarchy

Categories

- Primary Storage
- Secondary Storage
- Tertiary Storage

Primary Storage

- Can be operated directly by the CPU (Central Processing unit)

Example - main memory

Secondary Storage

- also called auxiliary, non volatile & used to store data and program for retrieval.

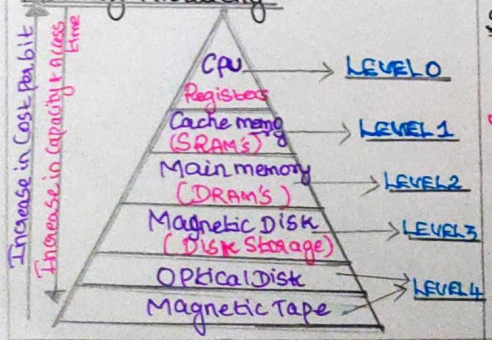
Example - Magnetic disk, Optical disk, Solid-state drive (SSD)

Tertiary Storage

- Removable disk, also called offline storage.

Example - CD-Rom's, DVD

Memory Hierarchy



Primary Storage level - Static Ram

- Cache memory (SRAM)
- Used by the CPU to speed execution of Program.

DRAM - work area for the CPU for keeping the data, also called main memory. (Dynamic RAM)

Disk Storage - between DRAM & magnetic disk storage.

- form of memory (flash memory)

Optical Drives - Capacity (4.5 to 15GB)

- Pre recorded data cannot be over written.
- CD-ROM, DVD's.

Magnetic Tapes - used for archiving & backup storage of data.

- Tape jukeboxes.

STORAGE ORGANIZATION

- two categories
  - Persistent data
  - transient data

⇒ data persist over long periods of time called Persistent

⇒ data accessed & processed repeatedly during the storage period. (transient)

SECONDARY STORAGE DEVICES

↳ Permanent loss of stored data from primary storage can be overcome by Secondary Storage

- Non-volatile Storage (Temporary memory)
  - Eg: Read only memory (Rom)
- Volatile Storage (Permanent memory)
  - Eg: Random Access memory (RAM)

RECORDS AND RECORD TYPES:

- Data stored in the form of records.
  - Record consists of values
  - value formed as 1 or more bytes
  - Record describes attributes and entities

EMP_NAME	SALARY
1234	5000

Attributes → EMP_NAME, SALARY; Entities → 1234, 5000

TABLE NAME: EMPLOYEE

DATA TYPES: Integer, float, long, String or characters [Fixed-length] [VARYING]

- Boolean [0 or 1] [true or false]

FIXED LENGTH RECORDS

↳ Setting a length

↳ Storing the records into the file.

* Storage when it is in fixed length called BLOCKS.

BLOCK SIZE - 4 to 8 KB (kilobytes)

- Record size exceeds, block size also increases.
- Block size increase problems like
  - i) Partial storing subpart of record
  - ii) more block access to read/write.
  - iii) Difficult to delete record

Overcome:

- Use FILE-HEADER
- file header carries information

Example - address of the first record

- Insert / Delete is easy in fixed length, because space freed by deleted record.
- Similar for insertion
- Challenges ⇒ fails for sorting the record

VARIABLE-LENGTH RECORDS

- Vary in size
- multiple blocks to store.
- Record type like arrays.

ISSUE:

- Representing single record for attribute extraction
- Defining the way of storing within a block.

REPRESENTATION

↳ numeric, values, dates etc.

↳ varchar (variable-character)

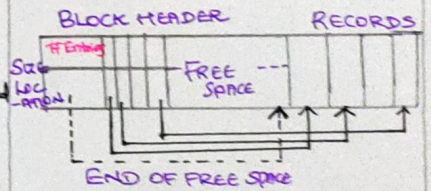
Initial part of record (offset, length)

- length of variable size attribute
- Place where records begins

Overcome

- Use Slotted-Page Structure.

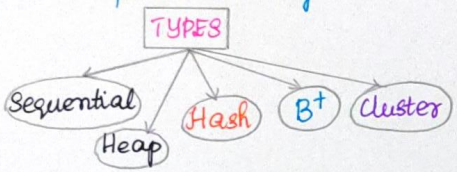
SLOTTED-PAGE STRUCTURE



- Records organized using slotted page structure.
- header in the starting of each block.
- header holds
  - No. of record
  - No free space remaining in the block
  - Location and size of records.
- Inserting & deleting method is efficient since no pointer directly point the record.

PRIMARY FILE ORGANISATION

- organizing data
- removes data redundancy
- improves accuracy



SEQUENTIAL FILE ORGANIZATION

- easy method
- new record: sequential order

HEAP FILE ORGANIZATION

- records added in memory
- not sorted

HASH FILE ORGANIZATION

- use hash function
- compute block address

B+ FILE ORGANIZATION

- key & index value of records
- tree like structure

CLUSTER FILE ORGANIZATION

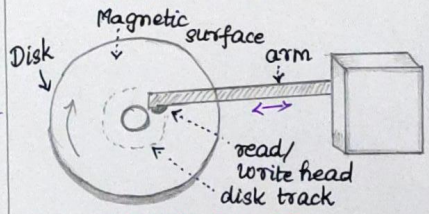
- combined data from multiple tables
- cluster: access data faster

SECONDARY STORAGE DEVICES

- store data for future use
- non-volatile
- eg: HDD, SSDs, optical disks
- persistent storage

(i) HARD DISK DRIVES (HDDs)

- magnetic disk
- placed vertically on spindle
- read/write head
- magnetized spot: 0's & 1's
- store large data
- concentric circles: TRACKS
- tracks divided: SECTORS



(ii) SOLID STATE DEVICES

- non-volatile
- stores data: flash memory

(iii) MAGNETIC TAPES

- Sequential access
- plastic ribbon: store data
- R/w speed: slower

ACCESS EFFICIENCY

- organize data
- I/O scheduling

OPERATIONS ON FILES

- open: read/write mode
- locate: file pointer
- read, write, close
- update
  - change data values
- retrieve
  - do not alter data

HEAP FILE

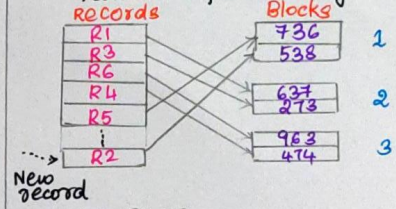
- unordered/pile
- simple, ordered as files
- new records: at end

OPERATIONS

1. EFFICIENT INSERTION
   - last disk copied to buffer
   - rewrite block to disk
2. LINEAR SEARCH
   - search block-by-block
   - avg searching: $(b/2)$ blocks
3. DELETION
   - find block
   - copy block to buffer
   - delete record from buffer

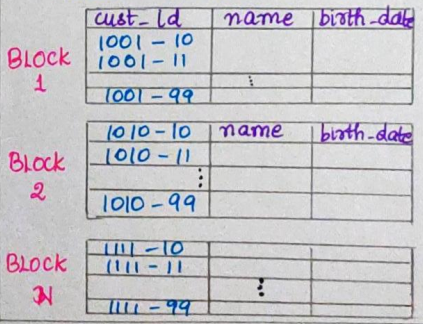
* use spanned/unspanned

- Variable/fixed length rec.



SORTED FILES

- ordered/sequential file



- ordering field: key field
- read records: efficient

ACCESS TIME

Type	Access	Avg. Access
Heap	Sequential	$b/2$
Ordered	Sequential	$b/2$
Ordered	Binary	$\log_2 b$

BINARY SEARCH

- sorted order key
- done on blocks

ALGORITHM

```
l ← 1; u ← b
while (u ≥ b) do
    i ← (l+b)/2
    read block i of the file;
if k < i then
    u ← i-1
else if k > i then
    l ← i+1
else if k = i then
    value = k
else k not found
```

QUICK SORT

- divide and conquer
- choose pivot element
- value < pivot: move left
- value > pivot: move right
- n log n comparisons

MERGE SORT

- divide and conquer
- divides list
  - two equal halves
- combine: sort them
- n log n comparisons
- better than quick sort

TOPIC: HASHING TECHNIQUES

HASHING
- search location of data
- eliminate index
- TERMINOLOGIES

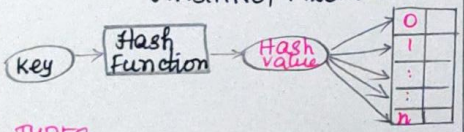
(i) DATA BUCKET
- memory location
- records stored

(ii) HASH FUNCTION (address)
- mapping function ($h(x)$)

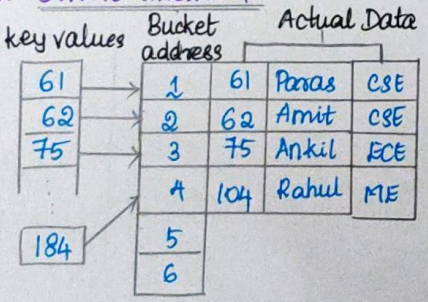
(iii) HASH INDEX
- prefix of hash value

(iv) KEY (k)
- attribute
- identify row

HASHING MECHANISM



TYPES
1. STATIC HASHING



- buckets count: constant
- ex: $h(x) = x \% 7$
  - for any 'x': value same
  * key: 107 → same address

3.

OPERATIONS
- Insertion
- Search
- Deletion

BUCKET OVERFLOW
- data exists in address
- COLLISION

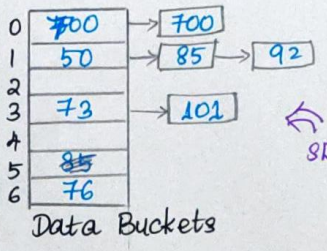
Collision Resolution Techniques
- Separate chaining (Open hashing)
- Open Addressing (Closed hashing)
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

SEPARATE CHAINING
- closed hashing
- same result - new bucket
- linked : previous one

LINEAR PROBING
* allocation
  - next free bucket
- open hashing

HASH FUNCTION: key mod 7

KEYS: 500, 700, 76, 85, 92, 73, 101



SEPARATE CHAINING

Data Buckets

Linear Probing - Eg:

76, 93, 40, 47, 10, 55

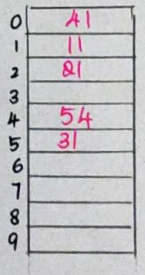
HASH FUNCTION: key mod 7

0	47
1	55
2	93
3	10
4	
5	40
6	76

$76 \% 7 = 6$

QUADRATIC PROBING
$h_1(x) = (h(value) + i^2) \% size$

eg: 11, 21, 31, 54, 41



$h(11) = 11 \% 10 = 1$

$h(21) = 21 \% 10 = 1$ =

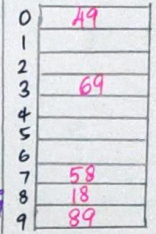
$(1+1^2) \% 10 = 2$

$h(31) = (1+2^2) \% 10$

$= 5$

DOUBLE HASHING
Hash function: key % 10

Insert 89, 18, 49, 58, 69



$h(89) = 89 \% 10 = 9$

$h(18) = 18 \% 10 = 8$

$h(49) =$ collision 9

$h(k, i) = (h_1(k) + i * h_2(k)) \% m$

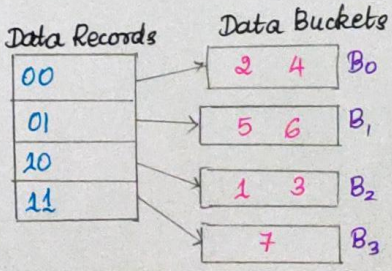
- uses secondary hash

DYNAMIC HASHING
- overcome bucket overflow
- data bucket: grow/shrink
- extendable hashing

Key	Hash address
1	11010
2	00000
3	11110
4	00000
5	01001
6	10101
7	10111

- prefix hash address

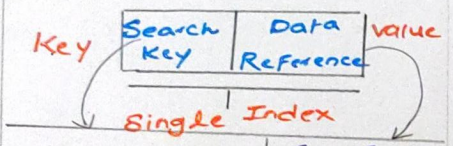
* last two bits 2 & 4 : 00 - B0
* 5 & 6 : 01 - B1



INDEX STRUCTURE FOR FILES

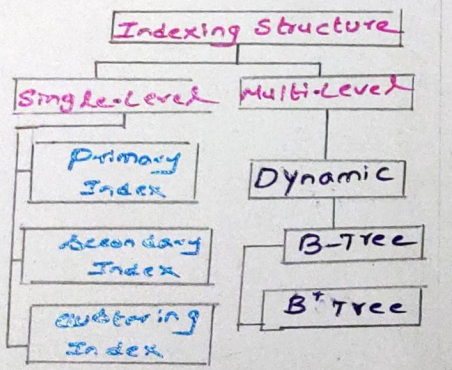
→ Data Structures to optimize the search time
→ Faster query Results
→ Quick data retrieval
→ Reduces disk to access data
→ consumes less Memory

INDEX STRUCTURE AND TYPE INDEX



Contains primary Key / Candidate Key to minimize query time

Set of Pointers which hold address of disk block



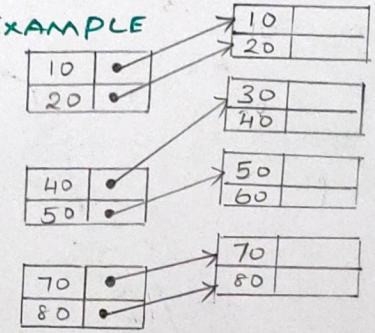
① SINGLE ORDERED LEVEL INDEXES

→ Ordered file
→ It follows pointer along with Field value
→ One index entry

a) PRIMARY INDEX

→ Primary Key values
→ No Duplicate values
→ Requires the row data to be ordered in Key index
→ One to one relation

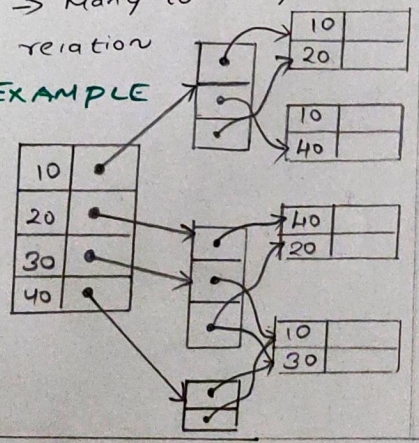
EXAMPLE



b) SECONDARY INDEX

→ Not a Primary index value
→ May have Duplicate values
→ Data May be in any order
→ Many to Many relation

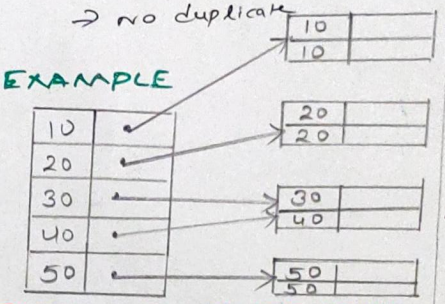
EXAMPLE



c) CLUSTERED INDEX

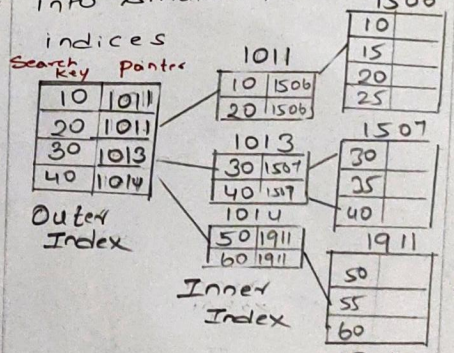
→ Data arranged in order (Dictionary)
→ only one way data stored
→ no duplicate

EXAMPLE



② DYNAMIC MULTI LEVEL INDEXES

→ Tree Structure
→ Node Formation
→ Parent and Child
→ Break down index into smaller parts of indices



B-Tree

→ Keys and record store in internal as well leaf node
→ No Duplicate keys
→ leaf not link
→ Search is not efficient and slower
→ Deletion of node is time consuming

B+ TREE

→ keys Stored in internal and records Stored in leaf node
→ contains Duplicate keys
→ leaf node linked
→ Search is very efficient and faster with less time

SEQUENCE

- User defined Object bound to Schema
- Generated Numeric values according to numeric value

SYNTAX at time OF Creation

```
Create Sequence Sequence_name
Start with initial value
increment by increment value
min value ...
max value ... Cycle/Nocycle
```

EXAMPLE

```
Create Sequence Sequence1
Start with 1
Increment by 1
min value 0
max value 100
Cycle;
```

Above Query will Create Sequence From 0 to 100

INDEX

→ Retrieve data From database
→ Quick and EFFICIENT
→ Duplicate value not allowed

SYNTAX

```
create index index_name ON
table_name (column1, column2
... column n)
```

EXAMPLE

```
create INDEX idx_fname ON
Persons (Firstname);
```

- Above index will be created on persons table

B-TREE, B+ TREE

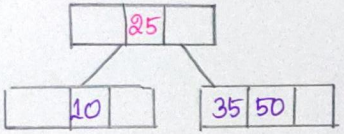
B-TREE

- self-balancing tree
- data: sorted form (stored)
- left < Root < Right Nodes

PROPERTIES:

- node: atmost 'm' children
- root: atleast 2 nodes
- leaf: same level

EG: B-TREE - ORDER: 4



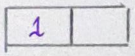
m = 4
max. keys = 4 - 1 = 3

OPERATIONS

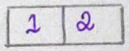
(i) INSERTION

order: 3 1 to 5

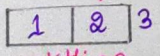
STEP: 1 insert (1)



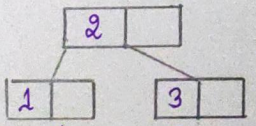
insert (2)



ins (3)

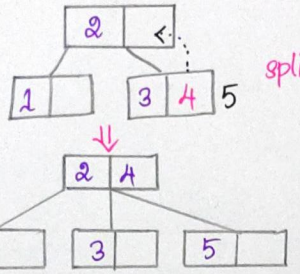


splitting



middle value ⟺ ROOT

ins (4) ins (5)



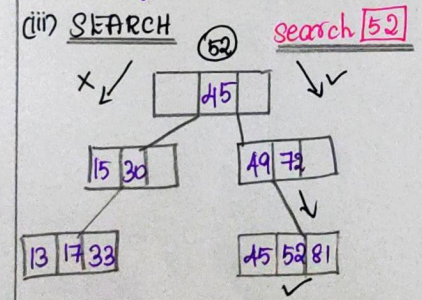
splitting

* LEAF NODE - EMPTY
  - add key value
  - ascending order
* LEAF NODE - FULL
  - split
  - middle value :- parent

(ii) DELETION

* 3 cases
  - delete leaf node
  - Internal node
  - key borrow

(iii) SEARCH



(i) element = first value
(ii) element - small - Left tree
(iii) element - big - Right subtree

B+-TREE

- extension of B-TREE
- records : LEAF
- LEAF: linked ⇓ makes search efficient

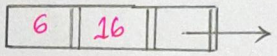
OPERATIONS

(i) INSERTION

B+tree order: 3 ⇓ 2 keys

6, 16, 26, 36, 46

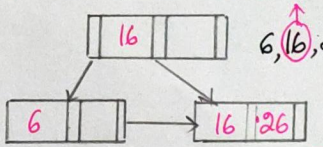
ins (6 & 16)



ins (26): overflow - split

first part: ceil (3-1)/2 : 6

second: 16, 26 COPY: 16

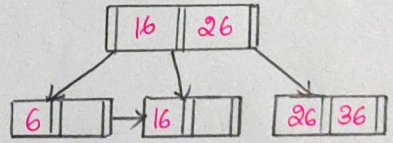


ins (36) 16, 26, 36

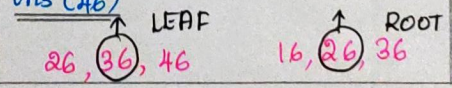
overflow - split

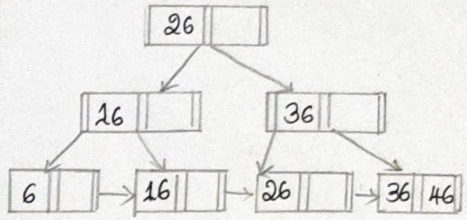
I part: ceil (3-1)/2 : 16

Part 2: 26, 36 COPY: 26



ins (46)





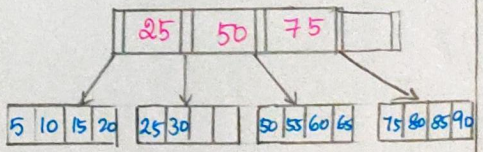
* LEAF NODE
  - no space : split
    copy middle node

(ii) DELETION
- delete key
- LEAF NODE
  - minimum elements

Case 1: - merge node
  - with siblings
  - delete key

Case 2: merge node with sibling
  - move key down

(iii) SEARCH



search 45, 15

(i) 45 - not found

(ii) 15 - < 25 - left subtree
  - position found

Query Processing

- processing high-level SQL queries via scanned, parsed and validated.

Steps

Query (SQL)
↓
Scanning, parsing & validating — identifies tokens, checks syntax, checks all attributes & meaningful form
↓
Immediate form of query — query graph
↓
Query optimizer — produce good execution plan
↓
Execution Plan
↓
Query code generator
↓
code to execute the query
↓
Runtime database processor — execute the compiled/interpreted code
↓
Result of query

Translating SQL Queries into Relational Algebra.

- SQL queries decomposed into query blocks
- Query blocks contains expression, SELECT-FROM-WHERE, GROUP BY, HAVING clauses
- Aggregate operators MAX, MIN, SUM, COUNT

Example

```
SELECT Lname, Fname FROM employee
  WHERE Salary > (SELECT MAX
    (salary) FROM employee WHERE
    Dno = 5);
```

⇓ decompose into 2 query blocks

(inner block)
```
SELECT MAX (Salary) FROM employee
  WHERE Dno = 5)
```
(outer block)
```
SELECT Lname, Fname FROM
employee WHERE Salary > c
```

Translated RA expression:

$\mathcal{T}_{MAX\ Salary}(\sigma_{Dno=5}(employee))$

$\pi_{Lname, Fname}(\sigma_{Salary > c}(employee))$

c - result returned from inner block.

Measures of Query Cost

Consider the resources;
- no. of disk accesses, CPU execution.

Query cost = (no. of seek operations x avg. seek time) + (no. of block read x avg. transfer time) + (no. of blocks written x avg. transfer time for writing).

Selection Operation Cost

Selection Condition	Cost
Full table scan	b
Linear search	b/2
Binary search	$\log_2(b)$
Primary index	x + 1
Primary index with multiple records	x + (b/2)
Clustering index	x + (s/bfr)
Secondary index	x + s

b - blocks; x - levels; f - factors

Example

```
SELECT * FROM employee WHERE
  empID = 123456789;
```

1. Cost of Full table scan = b_E = 2000 blocks.
2. Avg. cost Linear search = b/2 = 1000 blocks
3. Cost secondary Index on empID = x + 1 = 4 + 1 = 5 blocks

Sorting Operation

- ORDER BY clause is used
- uses a Sort-merge strategy
- Sorting phase - sorting small portions fit for buffer space
- Merging phase - merged one or more merge passes.

Sort cost = $(2 * b) + (2 * b * (\log_{d_M} n_R))$

Join Operation

Join Type	Cost
Nested loop join	$b_R + (b_R * b_S) + [js * \|R\| * \|S\| / bfr_{RS}]$
Index join	$b_R + (\|R\| * (x_B + s_B)) + [(js * \|R\| * \|S\|) / bfr_T]$
Sort Merge join	$b_R + b_S + [(js * \|R\| * \|S\|) / bfr_{RS}]$

Evaluation of Expressions

Query Evaluation → Materialization, pipelining

Demand driven ←
Producer driven ←

```
SELECT * FROM Student S,
Class C WHERE S.ClassID =
C.ClassID AND C.ClassName
= 'Design01';
```

Materialization
- queries broken into individual queries

π all column ∞ σ ClassName = 'Design01' (Class Table) → temp table
Student table

cost = individual select + temporary table

Pipelining
- each queries result pass to next queries

π all column ∞ σ ClassName = 'Design01' (Class table) → Pipeline result
STUDENT table

Estimating statistics of results

- No. of records in table, blocks, no. of unique columns
- Levels of index, length of record

Heuristic Query Optimization

- Rule based
- Systematic (cost Based)

- Query represented as a tree data structure.
- Operations at the interior nodes
- data items (tables, columns) are at the leaves.
- Query evaluated in a depth-first pattern.

Example: Query

Select Stuno, FName, LName from PROJECT, DEPT, STUDENT where StuNO = Stuid and Location = 'chn';

(OR)

Relational algebra

$\pi_{Stuno, StuNO, Lname}(\sigma_{Location='chn'}(\sigma_{StuNO = Stuid}(P \times DEPT)))$

To know which is more efficient rules are described below.

Rule 1: Cascading Selections (Conjunctive Conditions broken into individual)

Rule 2: Commutative of the selection operation.

Rule 3: Commuting Selection, Projection

Rule 4: Commutativity of join & cross Product

Rule 5: Associativity of union, intersection join

Rule 6: Logical transformation of Selection Conditions.

Steps in heuristic optimization

↳ Deconstruct the Conjunctive Selections.

↳ Move the selection operations down the query tree

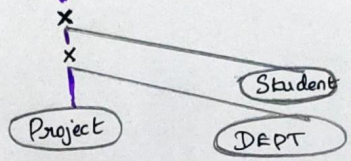
↳ Replace the Cartesian Product.

Example of Heuristic Query Optimization

1. Original Query tree

π Stuno, Stuid, LName

σ Stuid = StudNO $\wedge$ Location = 'chn'

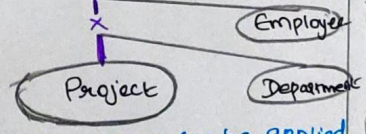


2. Use Rule 1 to Breakup Cascading Selections

π Stuno, Stuid, LName

σ Stuid = StudNO

σ Location = Chn



Similar all other rules can be applied

Systematic (Cost Based) Query Optimization

Cost components:

1. Access Cost to secondary Storage
2. Storage Cost
3. Computation Cost
4. Communication Cost

Overview (Cost Estimates in Query Optimization)

↳ Enumerate all the legitimate Plans $\{P_1, P_2, \dots P_n\}$

↳ set of operations $\{O_1, \dots O_k\}$

↳ Enumerate Access Routines

↳ Select the access routine to reduce the Cost.

IMPLEMENTATION OF DATABASE CONNECTIVITY

The data access methods and its types

- Open Database Connectivity (ODBC)
- Java Database Connectivity (JDBC)

ODBC:

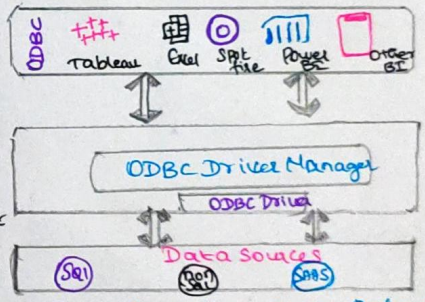
- developed by Microsoft

→ Application Program interface (API) for accessing data.

Adv: fast, SQL Enabled

Dis Adv: Not Portable, inflexible

ODBC ARCHITECTURE



→ ODBC Enabled Application Performs Processing by Passing SQL Statement

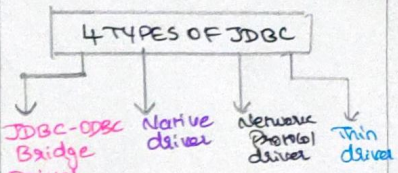
ODBC driver manager - Loads & unloads ODBC function calls

ODBC driver - Submit SQL to Specific data source

Data source - any file that run on application

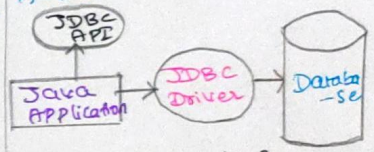
JAVA DATABASE CONNECTIVITY (JDBC)

- JDBC is a Java API to Connect and execute the Query with the database.
- JDBC API uses JDBC driver to Connect with the database.



⇒ JDBC API to access tabular data stored in Relational DB

⇒ Save, update, delete and fetch data.



JDBC 4.3

↳ Java.Sql Package Contains Classes & Interfaces

Java Connectivity with MySQL

Driver class: drive class for the mysql database is Com.mysql.jdbc.Driver

Connection URL:

jdbc:mysql://localhost:3306/Sonoo

localhost: Server

Mysql: database

Ipaddress: Running

Port: 3306

Sonoo: Database name.

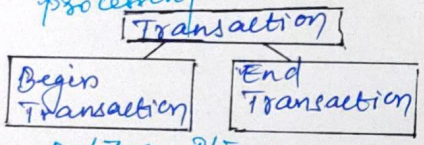
Username & password from root Sql

Sample:

```
import java.Sql.*;
Class MysqlCon{
public ..... {
try{ ..... .....
..... .....
}
```

Transaction Processing

* Concurrent Execution of transaction and recovery
* Single User Vs Multi User System
* Transactions, Database Items, Read and Write operations and DBMS Buffer
* Logical Unit of database processing

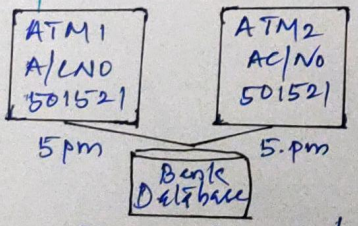


* Database Items
  read_item(X)
  Write_item(X)
  X - program variable

Need for Concurrency Control

* More than one transactions running simultaniously
* Inconsistency of database may occur

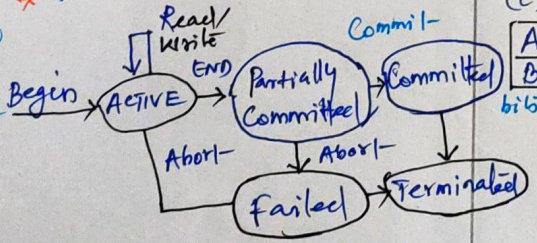
Example: In a joint bank account both persons withdraw Rs. 5000/- in different branches, it leads to database inconsistant state



* Temporary update
* may times in ATM leads to problem.
* Unrepeatable read problem
* Types of failures
  1. Computer failure
  2. Transaction / System errors
  3. Local error / execution condition detected.
  4. Concurrency control enforcement
  5. Risk failure

Transaction States

* Atomic unit of work
* BEGIN TRANSACTION
  Begining of transaction operation
* READ or WRITE - Operations on database
* END - TRANSACTION



* COMMIT TRANSACTION
* ROLLBACK (or ABORT)

Desirable Properties of Transactions

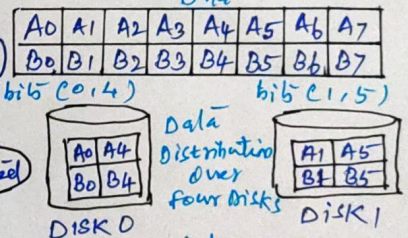
(1) Automicity - transactions in atomic unit of processing

ACID
(1) Atomicity
(2) Consistency
(3) Isolation
(4) Durability

(2) Executing from begining to end without interference
(3) Execution of one transaction not interfered by another transaction
(4) Changes will not affect the committed transactions

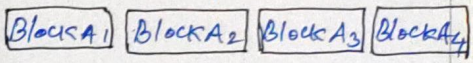
Disk Access using RAID Technology

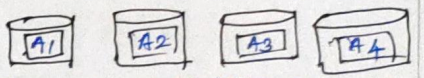
* Major advances in Secondary storage technology
* Levels - 0 to 6
(a) Data Striping
(b) Bit-level striping
(c) Block-level striping



Redundant Array of Inexpensive Disk (RAID)

* Data Striping distributes data transparently over multiple disks
* A byte is split and individual bits are stored as independent disks (Bit level striping)
* Block-level striping stripes blocks across disks
* Blocks are logically numbered from 0 in sequence. (0 to n-1)





Block-level striping across disks.

Improving Reliability with RAID:

* Mirroring (or) Shadowing
* Two identical physical disks that are treated as one logical disk

Improving performance with RAID:

* Disk array employ the technique of data striping - higher rate (data)
* Data can be read/written only one block at a time.

LEVELS of RAID

=> 0 uses data striping, no redundancy
=> level 1 - uses mirred disks
=> Level 2 - Memory style redundancy by using Hamming code
=> Level 3 - single parity disk
=> Level 4 & 5 - Block level data striping
=> level 6 - Read Soloman Codes.

SCHEDULE AND RECOVERABILITY

- Series of operation
- one transaction to another
- preserve order of operation

TYPES

Serial | Non-serial | Serializable

SERIAL SCHEDULE

- completely execute one transaction

T1	T2

read(A);
A:=A-m;
write(A);
read(B);
B:=B+m;
write(B);	

	read(A);
	A:=A+m;
	write(A);

NON-SERIAL SCHEDULE

- interleaving operations

T1	T2
read(A); A=A-m;	
	read(A) A:A+m;
write(A);	
read(B);	
	write(A);
B:=B+m; write(B);	

SERIALIZABLE SCHEDULE

- used to find non-serial schedule
- identifies schedule - correct
- non-serial schedule - result equivalent

TYPES OF SCHEDULES

SCHEDULES IN DBMS

- Serial
- Non-Serial
  - Serializable
    - Conflict Serializable
    - View serializable
  - Non-serializable
    - Recoverable
      - Cascading schedule
      - Cascadeless schedule
      - Strict schedule
    - Non-Recoverable

SERIALIZABILITY

- non serial : leads inconsistency
- identify non-serial

CHARACTERISTICS

- behave as serial schedule
- consistent

CONFLICT SERIALIZABLE

- swap nonconflicting operation
- convert non-serial - serial

OPERATIONS - CONFLICTING

- operations : different transaction
- same data item
- 1 operation : write

T1	T2
Read(A);	
write(A);	..... read(A);
read(B);	

conflicting operations

* W(1) & R2(A) - conflicting operations
  - same data item

VIEW SERIALIZABILITY

- view equivalent - serial schedule
- follow 3 conditions

1. initial reader - same

S1: T1: read(x); S2: T2: read(x);

2. W/R sequence : same

S1:

T1	T2
write(x);	
	read(x);

S2:

T1	T2
write(x);	
	read(x);

3. final writes - same

S1: T1: read(x); ... write(x); S2: T2: read(x); ... write(x);

NON-SERIALIZABLE SCHEDULES

- may or may not be consistent
- may/may not be recoverable

TYPES

IRRECOVERABLE SCHEDULES

- T2 performs dirty read
- commit before T1.

T1	T2
R(A) W(A)	
⋮	R(A) W(A) Commit
Rollback	

// Dirty Read

RECOVERABLE SCHEDULES

- T2 performs dirty read
- commit delayed / roll back

T1	T2
R(A) W(A)	
⋮	R(A) W(A)
Commit	
	Commit

// Dirty Read

// Delayed

TYPES OF RECOVERABLE SCHEDULES

CASCADING SCHEDULE

- Cascading Rollback / Cascading Abort
- failure : rollback

T1	T2	T3	T4
R(A) W(A)			
	R(A) W(A)		
		R(A) W(A)	
			R(A) W(A)
Failure			

- T1 fails, rollback: T2, T3, T4

CASCADELESS SCHEDULE

- committed read operations
- avoids cascading rollback
- uncommitted write operations

T1	T2	T3
R(A) W(A) Commit		
	R(A) W(A) commit	
		R(A) W(A) Commit

STRICT SCHEDULE

- neither read/write
- last transaction - written - committed/aborted
- more restrictions

T1	T2
W(A) Commit/ Rollback	
	R(A) W(A)

- more strict than casecadeless schedules
- ALL STRICT - CASCADELESS
- ALL CASCADELESS - NOT STRICT

CONCURRENCY CONTROL

- Controlling concurrent execution of the operation take part on a DB.

CONCURRENT EXECUTION

- Multi-user access at a time
- Simultaneous execution

PROBLEMS:

In database transaction 2 operation
- READ and write

If operation face problems in execution described as conflict.

Problem 1: Lost update problem (W-W conflict)

This problem occurs when 2 different transactions perform read/write operations on the same database.

Example:

TIME	Tx	Ty
t1	READ(A)	—
t2	A = A-50	
t3	—	READ(A)
t4	—	A = A+100
t5	—	—
t6	WRITE(A)	—
t7		WRITE(A)

Consider the above table where transactions Tx and Ty are performed on the same account A where the balance of account is $300

- At time t1, transaction Tx reads the value of Account A. (only read)
- At time t2, Tx deducts $50 from account A that becomes $250 (only deducted, not R/W)
- At time t3, t4, t6, t7 similar actions done and updated.

Due to various updates data becomes incorrect and inconsistent.

PROBLEM: 2

DIRTY READ PROBLEMS (W-R CONFLICT)

- Occurs when one transaction updates an item of the database.
- If transaction fails, rollback and new transaction started.
- Read/write conflict occurs between 2 transaction.

PROBLEM: 3

UNREPEATABLE READ PROBLEM (W-R CONFLICT)

- 2 different values are read for the same database item.

In order to maintain and manage execution protocol is introduced.

CONCURRENCY CONTROL PROTOCOLS

↳ ensures atomicity, isolation, Consistency, durability, Serializability.

3 Protocols

* Lock Based Concurrency Control Protocol.
* Time Stamp Concurrency Control Protocol
* Validation Based Concurrency Control Protocol.

Lock-Based Protocol

- Here any transaction cannot read/write data until it requires an appropriate lock on it.

2 locks

① Shared lock ② Exclusive lock

SHARED LOCK

- Also known as Read-Lock (only)
- data item can only read by the transaction
- Shared between the transaction
- transaction

EXCLUSIVE LOCK

- Data item can be both reads as well as written by transaction.
- Multiple transaction do not modify the same data simultaneously

FOUR TYPES OF LOCK PROTOCOLS

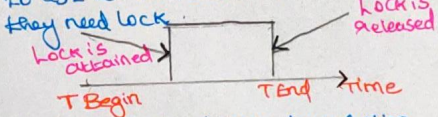
1. Simplistic lock protocol
2. Pre-claiming lock protocol
3. Two-phase locking (2PL)
4. Strict Two-phase locking (Strict-2PL)

SIMPLISTIC LOCK

- Simplest way
- allow all the transaction to get the lock on the data insert or delete.
- Unlock after completing the transaction.

PRECLAIMING LOCK

- Evaluate the transaction to list all the data items on which they need lock.

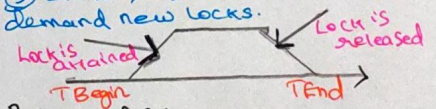


- Lock granted only transaction over
- rollback, wait until granted

TWO-PHASE LOCKING (2PL)

- 3 Parts

① Transaction execution starts, need permission for the lock.
② Transaction acquires all the lock,
③ Starts, after lock release, cannot demand new locks.



2 phase of 2PL

Growing phase - new lock on data, non can be released

Shrinking phase - existing lock held by the transaction may be released

STRICT TWO-PHASE LOCKING (STRICT-2PL)

- Similar to 2PL
- Difference is wait until whole transaction to commit, then release

TIME-STAMP ORDERING PROTOCOL

- Used to order the transaction based on their stamp (time).
- ascending order transaction.
- Priority of older transaction is higher, to determine the timestamp of the transaction logical counter is used.
- manage conflicts transactions.

WORKING PROCEDURE

TS(TI) - denotes timestamp of the transaction Ti
R_TS(x) - denotes Read time-stamp of x
W_TS(x) - denotes Write time-stamp of x

VALIDATION BASED PROTOCOL

- Also known as Optimistic Concurrency Control technique, 3 phases.

Start(Ti) - It contains the time when Ti started its execution.

Validation(Ti) - It contains the time when Ti finishes its read phase & start its validation

Finish(Ti) - It contains the time when Ti finishes its write phase.

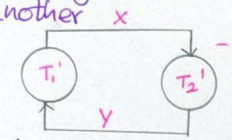
- This Protocol used to find time stamp for the transaction for SERIALIZATION using time stamp of validation phase.

Hence,

$$T(S) = \text{VALIDATION}(T)$$

- Serializability is determined using the validation process.
- Ensures degree of Concurrency during execution
- Contains transaction which have less number of rollback.

Deadlock
- waiting indefinitely for one another



- can't access items x & y.

Deadlock Prevention
- Two-phase locking
- Ordering all items

Transaction Timestamp (TS(T'))
- unique identifier assigned for each transaction.
- based on order started.

Two schemes -
(i) Wait-die - non preemptive tech.
* $TS(T_i) < TS(T_j)$ - T_i allowed to wait
- older trans. waits for younger trans. until available, abort T_j.

(ii) Wound-wait - preemptive tech
* $TS(T_i) < TS(T_j)$ - abort T_j, restart later

Transaction Without Timestamp
(i) No wait algo. - abort & restart after sometime
(ii) Caution waiting - reduce no. of needless abort/restart.

Deadlock Detection
- System checks deadlock state exists.

Wait-for graph - graph created based on transactions & locks.
- If graph has cycles/loops, deadlock detected.
- maintained by system for all transactions.

Timeouts - wait for system defined timeout period.

Starvation - when trans. can't proceed for indefinite period.

Timestamp-based Concurrency control
Timestamp - unique identifier created by DBMS.
(i) Increment each time by 1.
(ii) Current date/time

Timestamp Ordering Protocol
- trans. participate in serializable schedule.
- equivalent serial schedule only permitted.

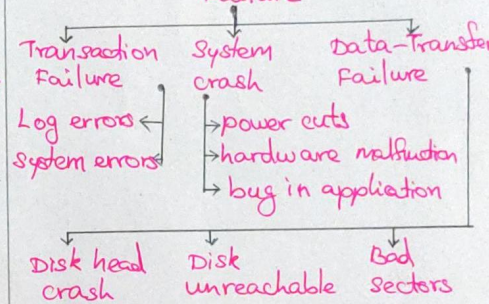
Two Timestamp values
* W_TS(x) - larger timestamp execute write(x).
* R_TS(x) - larger timestamp execute read(x).

Basic Timestamp ordering
- read & write executes in order.
- cascading rollback issue prevails.

Strict Timestamp Ordering
- ensure schedules are strict and conflict serializable.

Transaction Failure classification
- inability to execute transaction
- loss of data from DB.

Failure classification



Recovery Techniques
- possess fast data recovery
- heavily dependents on Sys. log
- Log contains trans. updates.

Storage recovery
* Log based recovery, sequence of records with all updates.
- contain fields; $< T_i, X_j, V_1, V_2 >$
T_i - Transaction identifier
X_j - Data item, V_1 - old value,
V_2 - New value.
* Undo and Redo operations
- system has old value prior to changes.

Recovery and Atomicity
- trans. atomic in nature
(i) Maintain logs for each trans.
(ii) Maintain shadow paging.

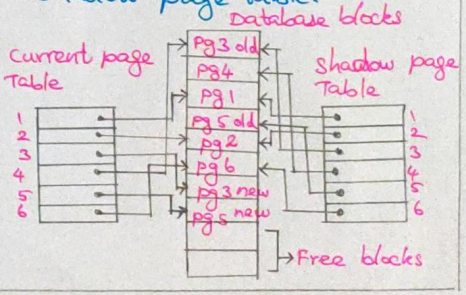
Buffer/cache Management
- data updated into main memory buffer, later to disk.
- collection of memory, managed using directory.
- dirty bit: 0 - buffer not modified
1 - buffer modified.

Recovery algorithms

(i) Deferred update (NO undo/redo)
- doesn't modify until partial commit.
- before commit, recorded in local workplace.

(ii) Immediate update (Undo/Redo)
- modification occur still trans. still alive.
- database modified immediately

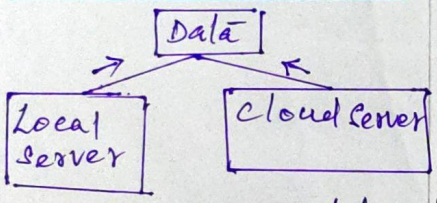
(iii) shadow paging
- provides atomicity & durability.
- pages - fixed logical storage
- page table - pages mapped to physical blocks.
- uses current page table and shadow page table.



MangoDB

* Document Database Non-Relational
* Stores data in flexible document
* Instead of table use Collections in MangoDB

Local Vs cloud Database

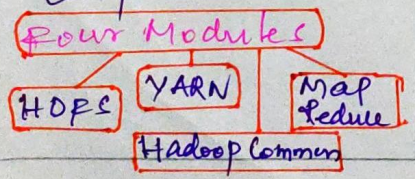


* Open source database
* cloud database platform

Hadoop

* Open source framework
* Store and process large amount of data sets
* clustering Multiple Computers

Four Modules

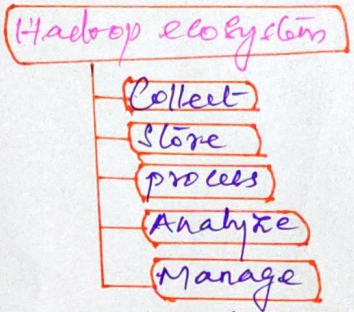


1. Hadoop Distributed File System (HDFS)
   => Better data throughput
   => High fault tolerance.
2. Yet Another Resource Negotiator (YARN)
   => Manage and Monitor Cluster needs
3. MapReduce
   => framework for parallel computation on data
   => Convert input data to data set
   => Reduce task and Aggregate output.
4. Hadoop Common
   => Common java libraries
   => Across all modules

How Hadoop Works

* Cluster servers manage storage and processing capacity.
* Distributed processing against huge amount of data
* Collect data in various formats
* API operations
* Name Node * Data Node

Hadoop ecosystem



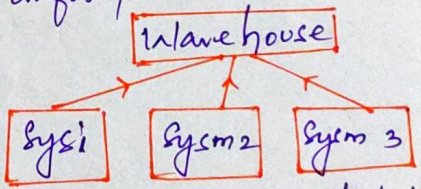
popular Applications
* Spark * presto * Hive

Hive

* Distributed fault tolerant data warehouse system
* Read, Write and Manage peta bytes of data
* process large data set
* HiveQL (like SQL)
* Hive queries into map reduce.

Data Warehouse

* Stores Highly structured information



* DB systems connected to Warehouse

Charactristics

* Stores large amount filtered and aggregated data.
* pre-defined and fixed relational schemas

Examples
Amazon Redshift
Google Bigquery
IBM - DB2 Warehouse

Lake:

* Repository of data like data warehouse
* Analyze the data to gain insites
* Store structured, semi structured and unstructured data.

Example: AWS S3
Google Cloud Storage
MangoDB Atlas Database
* Amazon RDS

Casandra

* Distributed Database from apache, highly scalable, large amount of structured data.
* No-SQL data base
* Handle Huge amount of data
* Database and linux flavours required to learn