SIMATS
School of Engineering

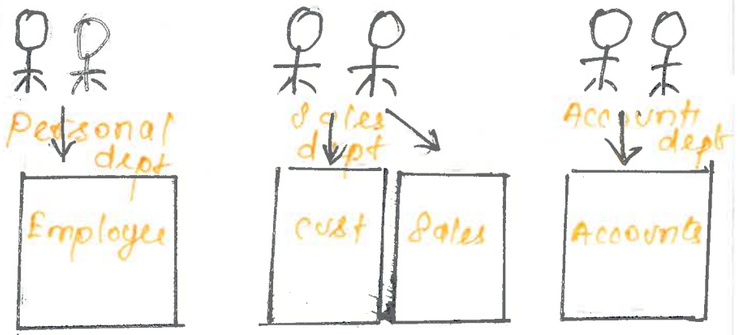
DATABASE MANAGEMENT SYSTEMS

Computer Science Engineering

Saveetha Institute of Medical And Technical Sciences,Chennai.

DBMS – INTRODUCTION

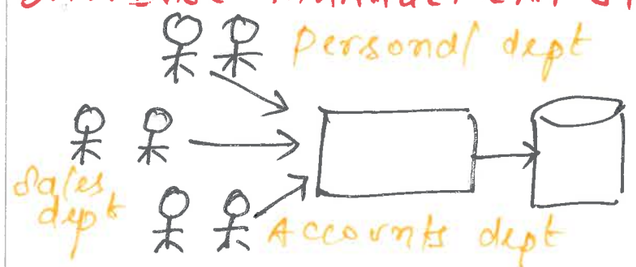
FILE MANAGEMENT SYSTEMS



DISADVANTAGES

→ Data redundancy and inconsistency
→ Difficulty in accessing data
→ Data Isolation
→ Integrity problems
→ Atomicity of updates
→ Concurrent access of data
→ security problems

DATABASE MANAGEMENT SYSTEMS



Data → raw fact

Information → processed data

Database → Collection of interrelated data

DBMS - Software to define, create and manipulate db

(e.g) mysql
Oracle
Sql Server
IBM DB2

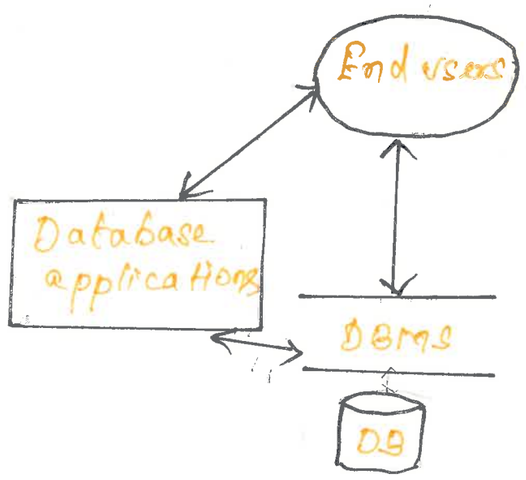
ADVANTAGES

→ Easy retrieval
→ Minimum redundancy
→ Data integrity
→ Data security
→ Reduced development time
→ Concurrent access
→ Data consistency

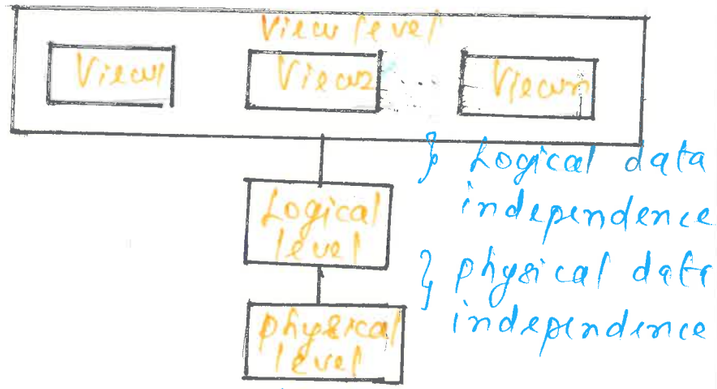
DISADVANTAGES

→ Larger in size
→ except mysql, costly

COMPONENTS OF DBMS



DATA ABSTRACTION (VIEWS OF DATA)



physical level – how the data stored, complex low-level data structures

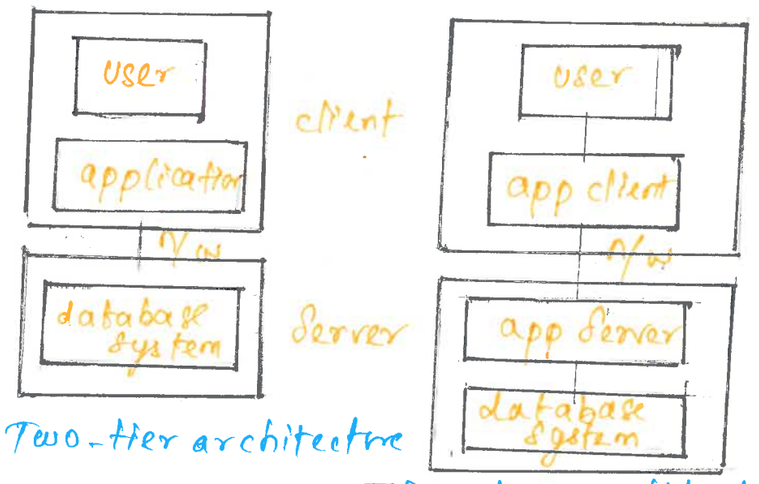
Logical level – what data stored, relationship

view level – part of the database.

SCHEMA – Overall design

INSTANCE – Instance data

DATABASE APPLICATIONS



Two-tier architecture

Three tier architecture

HISTORY OF DATABASE SYSTEMS

1960 – Charles
1970 – IBM's IMS
1976 – ER MODEL
1980 – RELATIONAL MODEL
1985 – OBJECT DBMS
1991 – MS-ACCESS
1995 – INTERNET DB
1997 – XML

CONCEPT MAP 2 - DATABASE SYSTEM STRUCTURE AND SOFTWARE ARCHITECTURE OF A TYPICAL DBMS

DATABASE SYSTEM STRUCTURE

- modules
- functional components
  - Storage manager (SM)
  - Query Processor (QP)

- SM - large - Storage Space
- Sample
  - i) gigabytes
  - ii) tetrabytes
  - iii) Megabytes
- Stored - disks
- Data - disk Storage - main memory
- QP - Access data
- Updates - Queries - Operations

STORAGE MANAGER

- interfaces - low level - application - Program
- file manager - DML -
  - STORING
  - RETRIEVING
  - UPDATING

SM COMPONENTS

- Authorization & Integrity manager
- Transaction Manager
- File manager
- Buffer Manager

PHYSICAL SYSTEM IMPLEMENTATION

→ Data file - Store
→ Data dictionary - meta data - Structure - Schema - database
→ Indicies - fast access - values

Fig. SYSTEM STRUCTURE

QUERY PROCESSOR

→ COMPONENTS

DDL INTERPRETER
- interprets - ddl statements - records - data dictionary

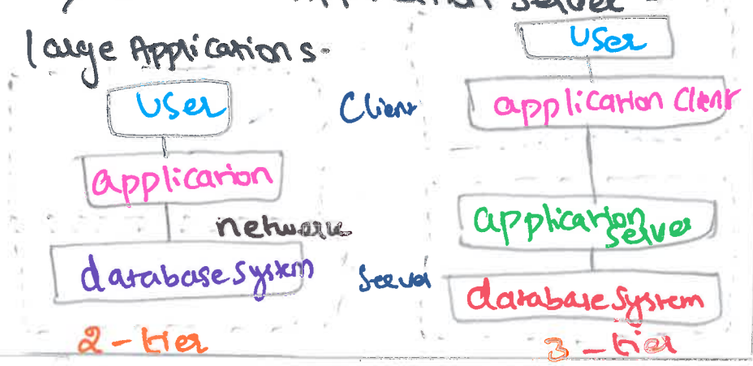
DML COMPILER
- translates - DML Statements - query language - evaluation plan
- 2 level instructions → low-level, query evaluation

QUERY OPTIMIZATION
- low cost - evaluation plan

QUERY EVALUATION ENGINE
- executes - low level - instructions - DML Compiler.

APPLICATION ARCHITECTURE
- Networks - Client - Server
- 2 parts ⇒ two-tier architecture
  ⇒ three-tier architecture

⇒ 2 tier - Application Program interface - ODBC - JDBC
⇒ 3 tier - Application Server - large Applications.



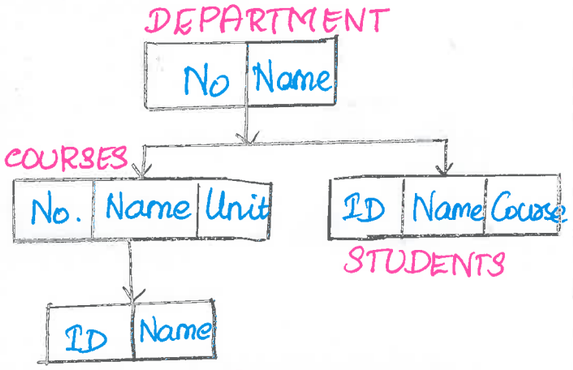
DATA MODELS

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

TYPES:

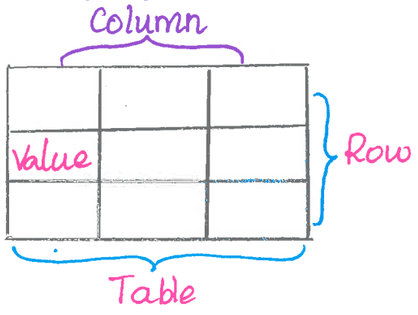
① HIERARCHICAL MODEL

* data - hierarchical tree structure



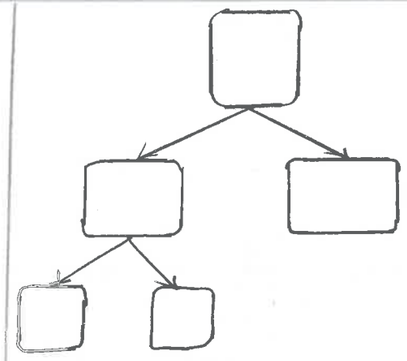
② RELATIONAL MODEL

* 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.

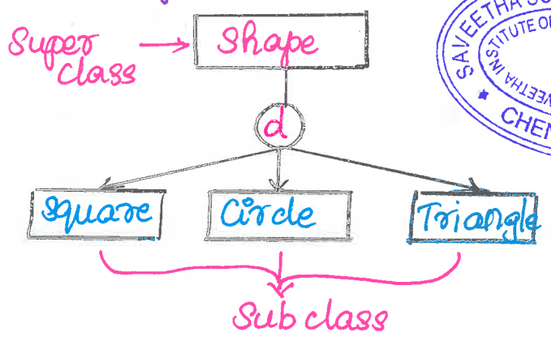
ENHANCED ENTITY RELATIONSHIP MODEL (EER)

* incorporates extensions to ER model.
* represents following concepts

① Subclass & Superclass

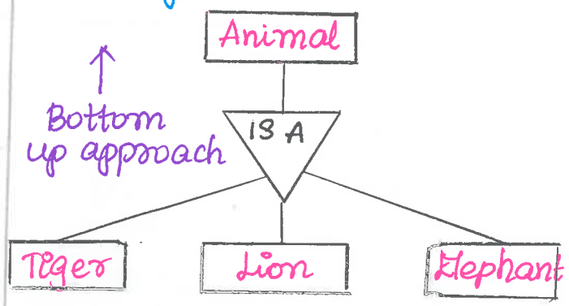
- concept of inheritance.

ⓓ symbol.



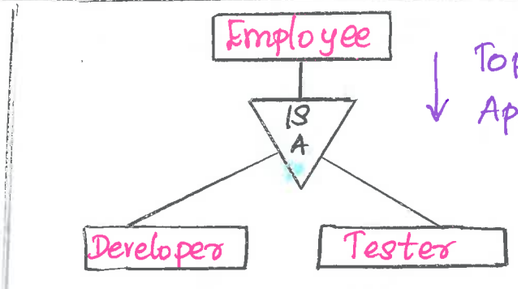
② Generalization

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



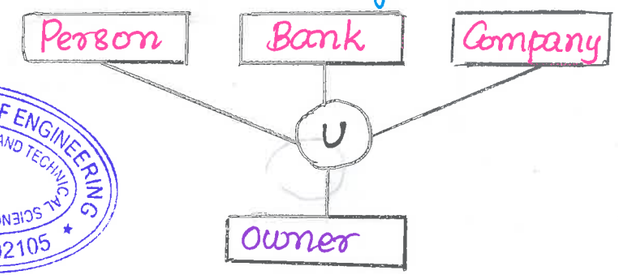
Specialization

- top down approach



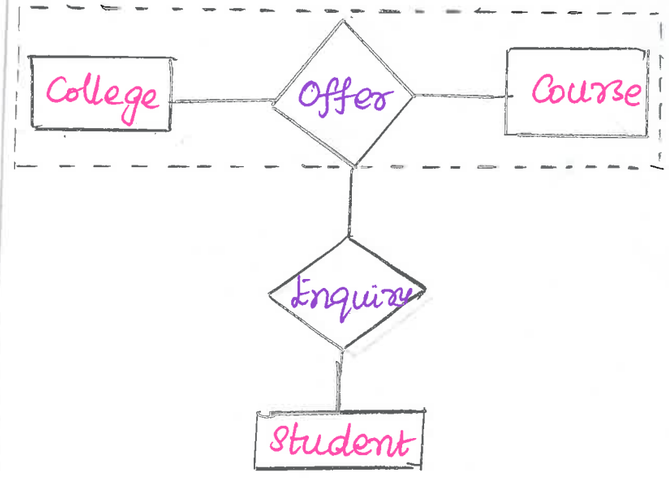
③ Category or Union

* 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

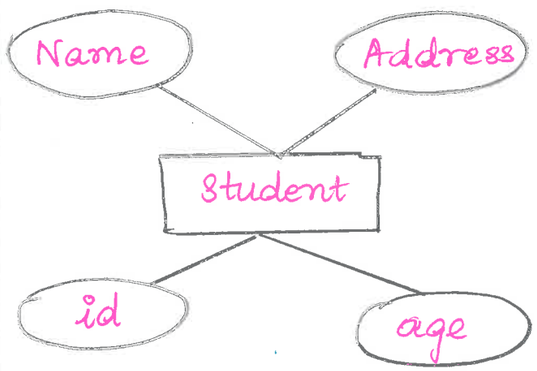
attributes

address, name, id, age.

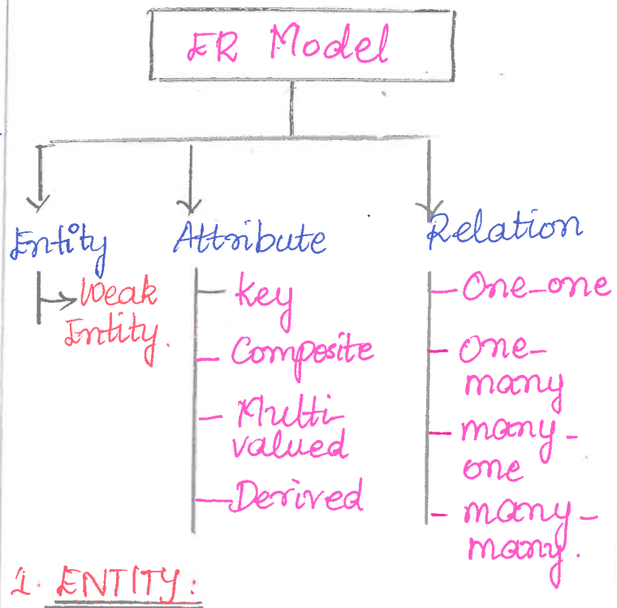
Entity: address

attributes

city, street name, pincode



Components of ER diagram

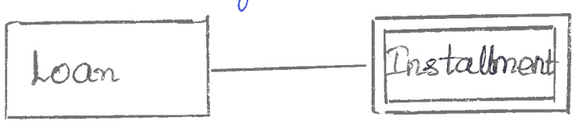


1. ENTITY:

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

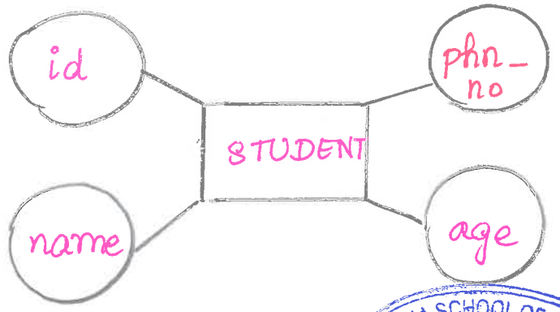
a) WEAK ENTITY.

* 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 characteristics of an entity.
* primary key.
* ellipse with text underlined
* eg: id.

b) COMPOSITE ATTRIBUTE:

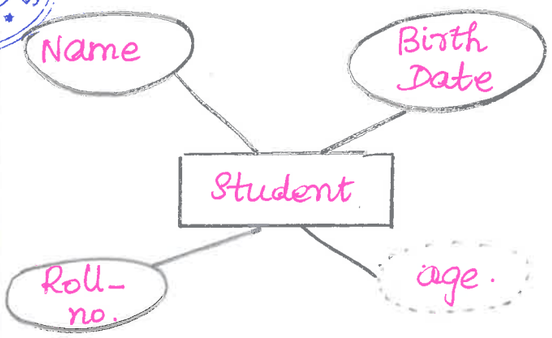
* composed of many other attributes.
* ellipse.

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.

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 (X)
* 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 (⟖)
* 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

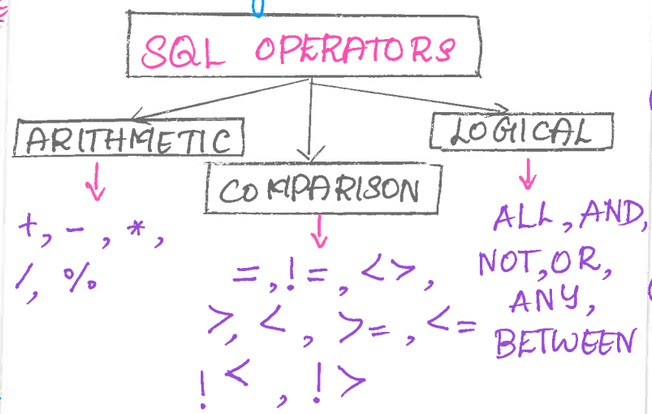
(iii) SAVEPOINT : roll back transaction back to a certain point

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 col 1 = value 1, col 2 = value 2
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.

NORMAL FORMS IN DBMS

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		TABLE 2	
STUD_NO	COURSE_NO	COURSE_NO	COURSE_FEE
1	C1	C1	1000
2	C2	C2	1500
1	C4	C3	1000
4	C3	C4	2000
4	C1	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 super key.
  - 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}
↓ decomposed 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

FUNCTIONAL DEPENDENCIES

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 form State r of 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 (of R)
legal extension of R.

Attributes
{State, Driver_license-Number} → SSn

SSn - hold 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.

↳ 2 Relation schema under 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($R_1 \cap R_2$) → ($R_1 - R_2$)

(or)

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

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 Parts to use relations.

TOPIC TITLE: Record Storage and Primary File Organization, Secondary Storage devices.

Record storage

* Data Collection
* Computerized database
* Stored physically.

Two Categories

(i) Primary storage

- Directly operated
- Main memory
- Fast access
- Limited storage.

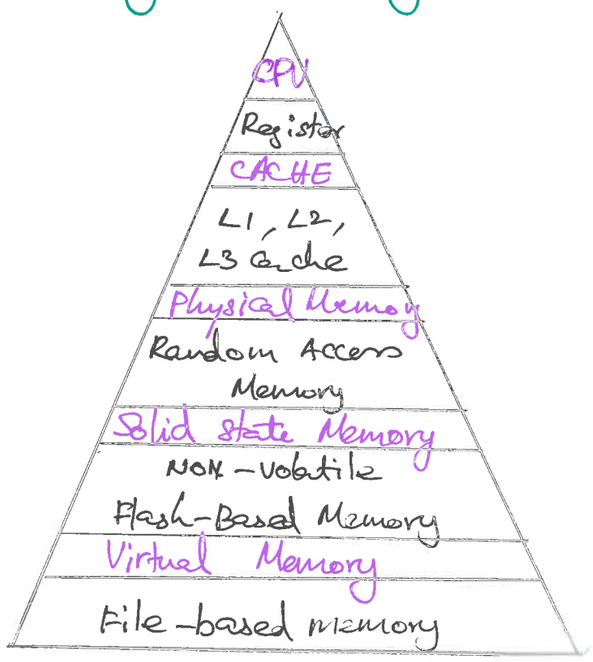
(ii) Secondary storage

- Magnetic disks,
- optical disks
- Tapes
- Large capacity
- Low cost
- slow access
- No direct access

(iii) Tertiary storage

- Removable media
- CD-ROM's, DVD's
- Larger Capacity
- Cost Less
- Slower access

Memory Hierarchy



Storage organization

(a) Persistent Data

- Store large data
- Persist long time

(b) Transient Data

- Persist short time
- For execute programs.

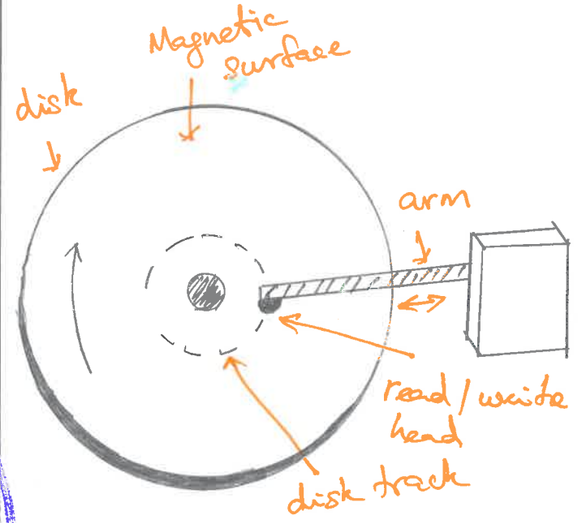
Secondary storage

(i) Hard Disk Drive (HDD)

- Magnetic disks
- Store large data
- Format 0's & 1's
- bytes/characters.
- Single-sided - one side data
- Double-sided - both side data
- Assembled - Disk pack.
- Circle as tracks
- same tracks - cylinder
- Blocks / sectors - tracks divided

(ii) Access efficiency

- Data buffering
- Organize data
- IO scheduling



- Log disks
- Ahead read
- Use SSD/Flash

(iii) Solid state Devices

- Flash Memory based
- intermediate
- Non volatile

(iv) Magnetic Tapes

- Sequential access
- mounted
- scanned
- Backup/archive

TOPIC TITLE : Operations on Files, Heap File, Sorted Files

Operations on Files

1. Retrieval operation
   - Data no change
2. Update operation
   - Data change
   - insert, delete

Accessing Commands

- Open
- Reset
- Find
- Read
- FindNext
- Delete
- Modify
- Insert
- Close
- scan
- FindAll
- FindOrdered
- Reorganize

Program variables

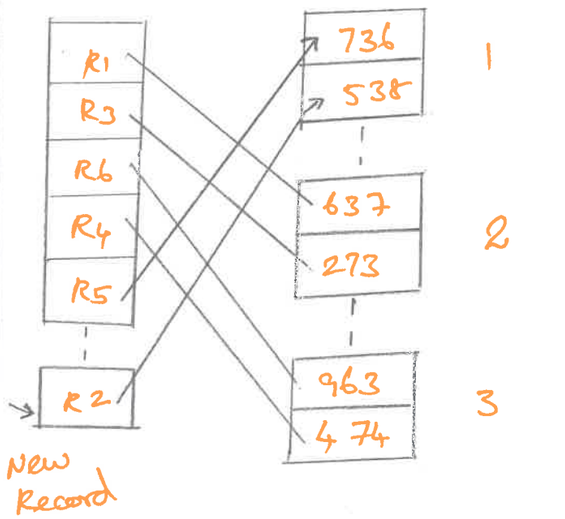
record-at-a-time

set-at-a-time

File Organization	File Access
- Physical placements - Files into records - Pages on secondary	- data retrieval - steps to read - Sequential accen - Back to end - Direct accen

Heap file

(i) Unordered
- unordered / Heap / pile
- insert at end
- order of insertion
- Efficient insertion
- Linear searching - Search
- reading as sorted
- Deletion Techniques
  - Rewrite
  - use deletion



Sort files

(ii) Ordered
- Ordered / Sequential
- sorted records
- insertion expensive
- Binary search used
- Efficient sorted
- Easy finding next

Access Time

Type	Access	Avg Access
Heap	Sequential	b/2
ordered	sequential	b/2
ordered	Binary	$\log_2 b$

(iii) Quick Sort
- identify pivot
- less pivot - move left
- greater pivot - move right
- n log (n)
- less stable

(iv) Merge sort
- Larger tables
- better than quick
- upto 4 records

Binary Search Algo.

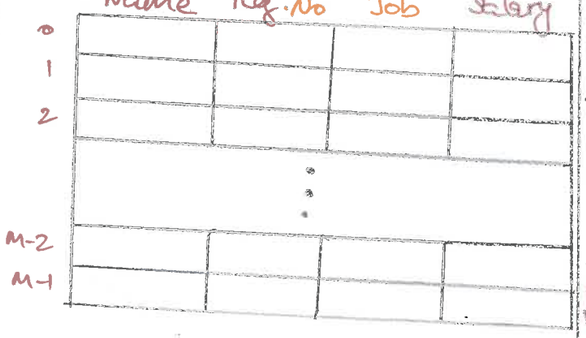
```
l ← 1; u ← b;
 while (u ≥ l) do
   begin i ← (l+u) div 2;
   read block
if k < (order)
   then u ← i - 1
else if k > ordering
   then l ← i + 1
else if k = value
   then goto found
end;
goto not found;
```

TOPIC : Hashing Techniques

Hashing

- Fast access
- search conditions
- Single equality condition
- hash field value h
- yields address

(i) Internal hashing



- Hash table used
- Array of records
- Index : 0, 1, 2, ... M-1.
- transforms hash field
- integer b/w 0 to M-1.

$h(K) = K \bmod M$

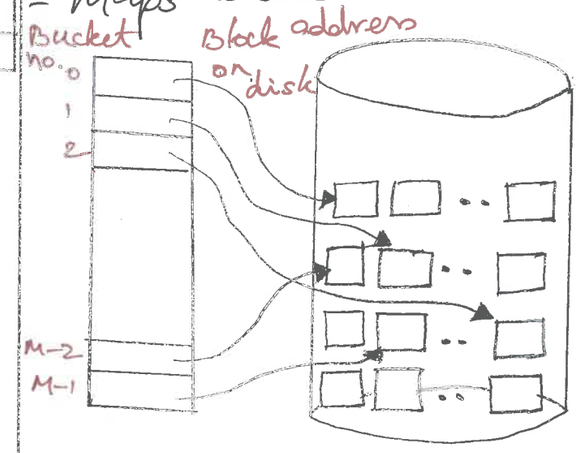
↳ hash key field

Problem

- hash field >> address
- doesn't guarantee
- collision occurs
  - open addressing
  - Chaining
  - Multiple hashing

(ii) External hashing

- made of buckets
- each has multiple records
- either single / cluster
- maps bucket numbers



- collision less
- Static hashing
  - fixed buckets

Hashing with Dynamic file

(a) Extendible hashing

- performance doesn't degrade
- used in dynamic files
- file grows (dynamic)
- stores access structure
- Binary representation
- Directory
  - 2^d bucket address
  - d - depth
  - d = 1

(b) Linear hashing

- expands
- shrinks
- no directory
- File M : 0, 1, 2, ... M-1.

$h_i(K) = K \bmod M$

- collision leads overflow
- split bucket.

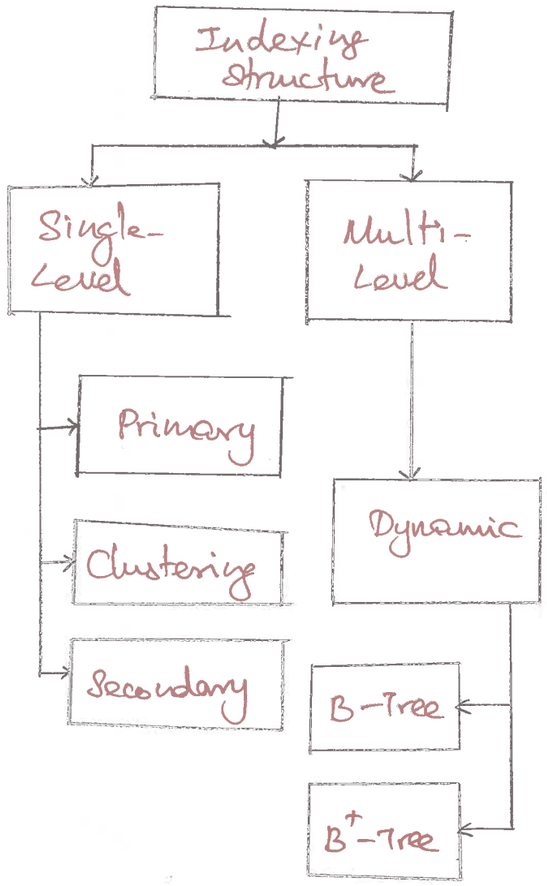
start - 0 ; end - M.

(c) Dynamic hashing

- Tree - structure directory
- add / remove
- on demand
- use large no. of values

Order Indexing	Hashing
- Primary key	- hash fun. value
- poor in data increases	- best in addition
- slow for insert / delete / update	- maintenance costlier
- used for limited range	- retrieve search key
- many unused blocks	- memory managed
- unused can't reused	- overflow handled.
- maintenance required	

Topic : Index Structure for files, Different types of Indexes



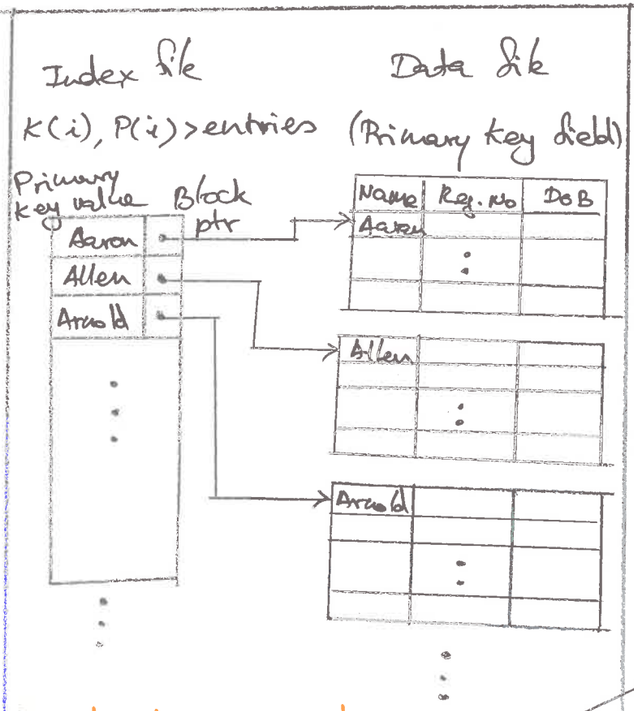
- Speed up retrieval
- use search conditions
- secondary access path
- any field - Index
- multiple indexes used
- Index ordered files
- Tree data structure used.

(i) Single-Level Ordered Indexes

- textbook index
- use fields (attributes)
- stores with pointers
- blocks contains records
- pointer maps blocks
- ordered values

(a) Primary Index

- ordered records
- Two fields
  - Primary key K(i)
  - Pointer P(i)
- Single index entry
- dense / sparse
- pointer maps each block.

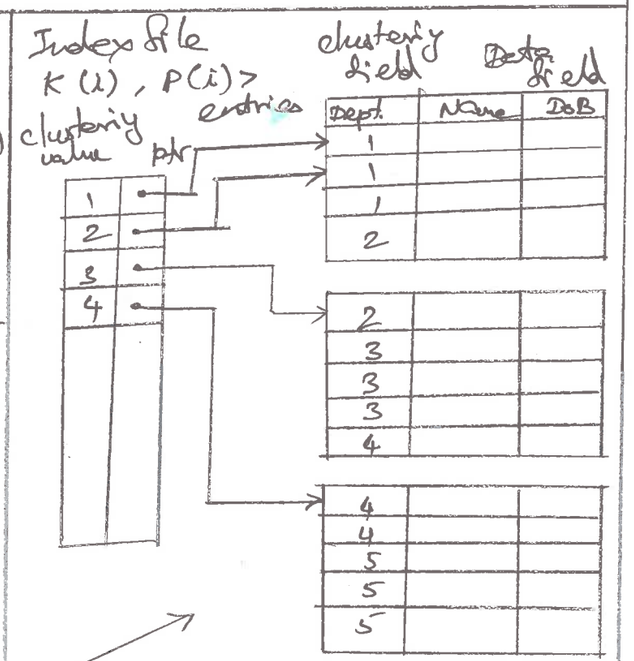


(b) Clustering Index

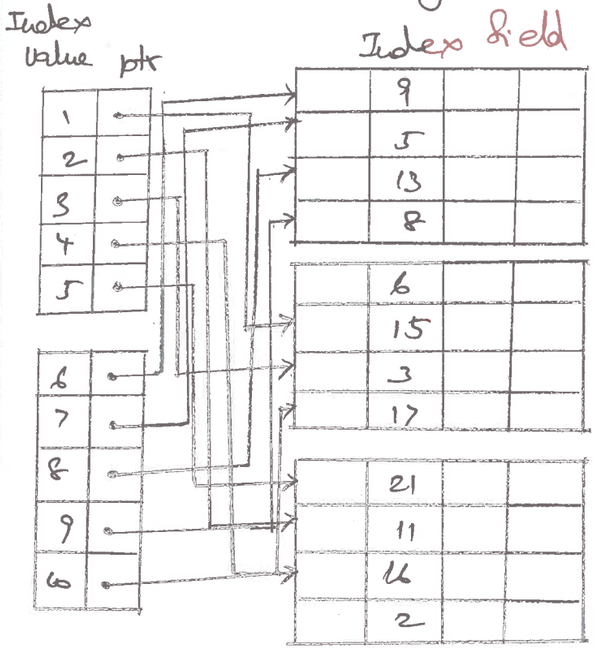
- Clustering field
  - physically ordered
  - non-key field
  - no distinct values
- Two fields used
  - clustering field
  - disk pointer

(c) Secondary Index

- secondary access
- primary access exists
- 2 fields to order



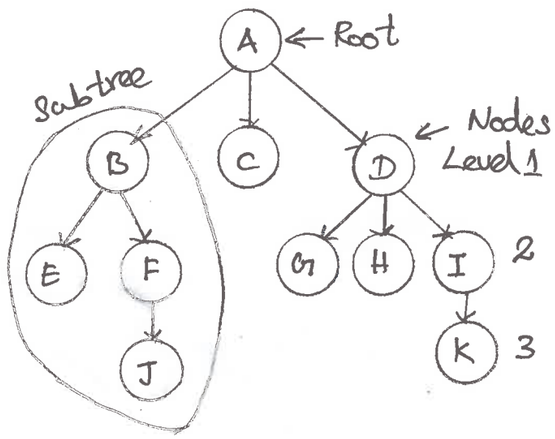
- more storage
- Long search time
- Slow than primary



TOPIC : B-Tree, B^+-Tree, Index, Sequences

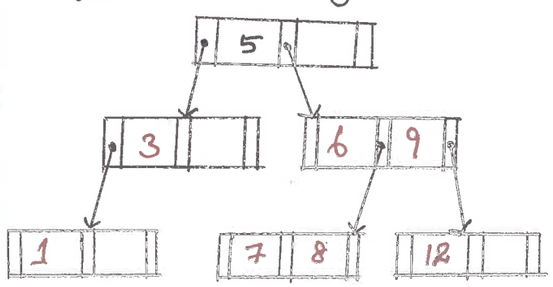
Dynamic Multi-Level Indexes

- Tree data structure
  - formed of nodes
  - has parent & child
  - Leaf has no child
  - ⇓
  - Internal Node
  - has descendent nodes



Search Trees

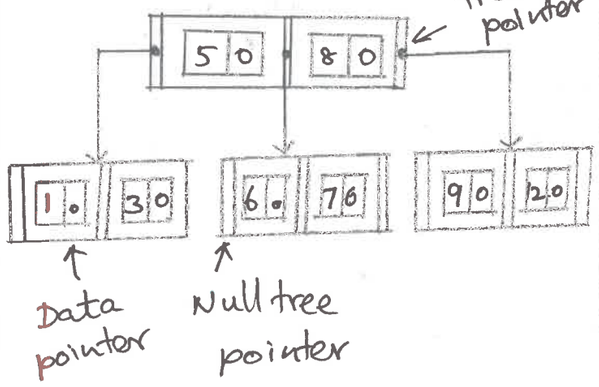
- guide record search
- for inserting & deletion



Search order p=3

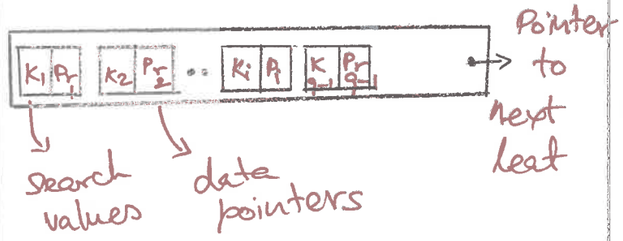
B-Trees

- provides multi-level access
- balanced tree
- less wastage space
- each node half-full.
- p-1 search values.



B^+-Trees

- data pointer at leaf nodes
- have entry
- point to record → key field
- point to data file → Non-key field
- Internal nodes
  - guide search
- search values } q-1
- data pointers }



Database objects

Object	Description
Table	Rows & Columns.
View	Logical representation
Sequence	generate primary key
Index	Improve performance
Synonym	object

Sequence

- Automatic sequence unique no.
- Sharable objects
- create primary key
- replace application code
- speed up efficiency.

Create Sequence

```
CREATE SEQUENCE dept_deptno
    INCREMENT by 1
    START with 91
    MAXVALUE 100
    NOCACHE
    NOCYCLE;
```


Index

- is a schema object
- speed up retrieval
- reduce disk I/O
- independent from table
- Maintained by oracle.

Create Index

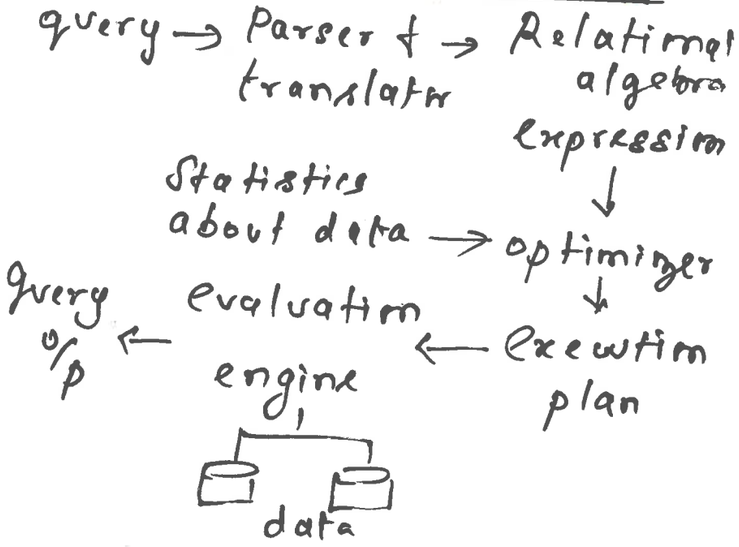
```
CREATE INDEX emp_ename_idx
    emp(ename);
```

- automatically defined by primary key
- unique index create.

QUERY PROCESSING

→ Steps in query processing
→ measures of Query cost
→ Selection operation
→ Sorting
→ Join operation
→ Other operations
→ Evaluation of expressions

Steps in query processing



Measures of query cost

Factors → disk access
→ CPU
→ network communication

CPU cost → response time
Disk cost → number of seeks
→ number of block reads
→ number of blocks written
→ average-seek-cost,
average-block-read-cost
average-block-write-cost

Cost for block transfer
$= b * t_T + S * t_S$

selection operation

→ index scan
→ linear search
→ clustering index, equality on key
→ clustering index, equality on non-key
→ secondary index
→ clustering index
→ conjunctive selection
→ disjunctive selection
→ negation

Sorting

→ external sort-merge

Join operation

→ Nested-loop join
→ Block nested loop join
→ indexed nested_loop join
→ Merge-join
→ Hash-join

Other operations

→ Duplication elimination
→ Projection
→ Aggregation
→ Set operations
→ Outer join

Evaluation of expressions

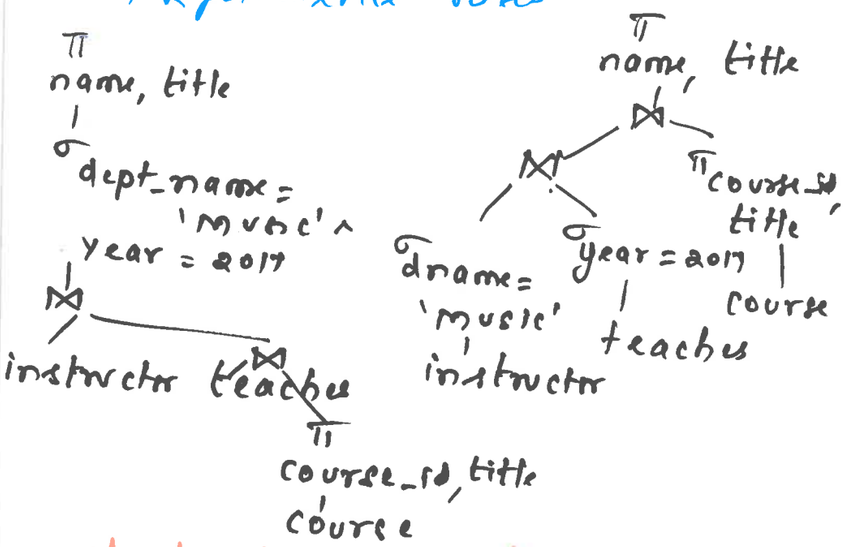
→ materialization
→ Pipelining

QUERY OPTIMIZATION

→ Transformation of relational expression
→ cost-based optimization

Transformation of relational expression

→ Equivalence rules

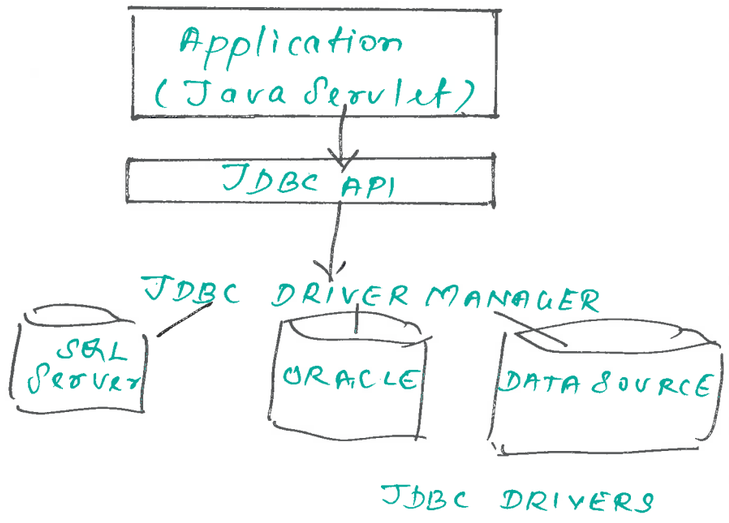


cost-based optimization

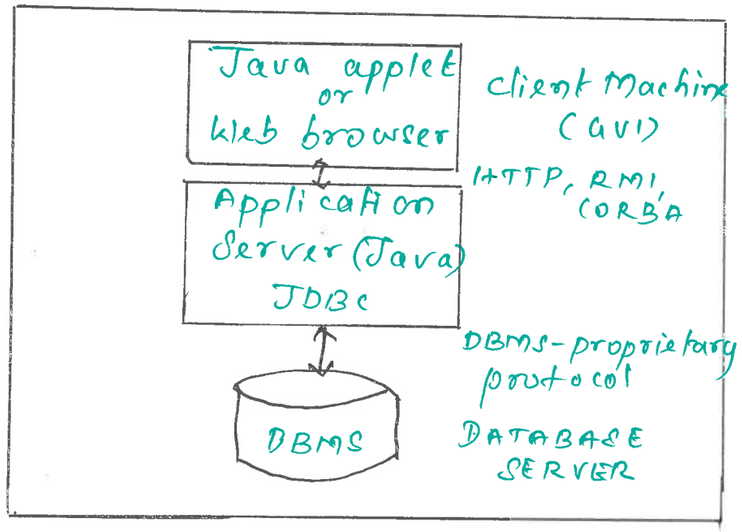
→ Need statistics of relations
→ Need statistics of expression results

IMPLEMENTATION OF DATABASE CONNECTIVITY

INTRODUCTION TO JDBC (JAVA DATABASE CONNECTIVITY)



JDBC ARCHITECTURE



ACTIVITIES
- → Connect database
- → Send queries
- → Retrieve results

JAVAAPI
- → java.sql
- → javax.sql

TYPES OF JDBC DRIVER
- → Type 1 - JDBC ODBC bridge
- → Type 2 - Native API connection driver
- → Type 3 - Network connection driver
- → Type 4 - Database protocol driver

STEPS TO CREATE JDBC APP

Loading driver
(Class.forName(driver)
↓
Establishing connection
(con = DriverManager.getConnection (url+db, user, pass)
↓
Executing statements
(res = St.ExecuteQuery("Select * from tablename)
↓
Getting result
↓
closing database connections

TYPES OF STATEMENT OBJECT

→ Statement : access database

```
Statement stmt = null;
try {
stmt = conn.createStatement();
}
catch() e { }
```

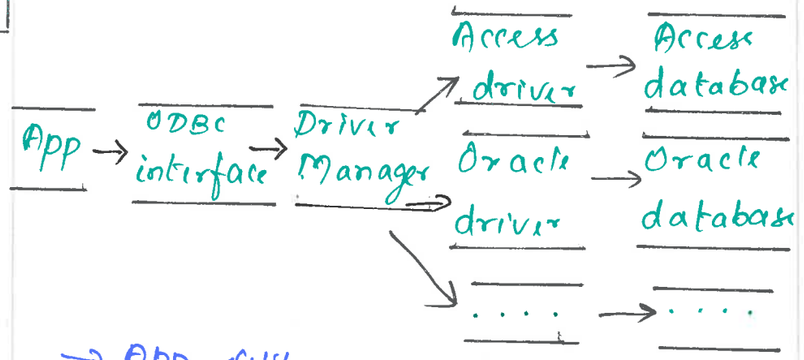
→ Prepared statement : use SQL statements many time

```
String SQL = "Queries"
PreparedStatement pstmt = null
pstmt = conn.prepareStatement(SQL)
```

→ Callable statement : access database stored procedures

```
CallableStatement cstmt = null
string SQL = "{queries}"
cstmt = conn.prepareCall(SQL)
```

ODBC (OPEN DATABASE CONNECTIVITY)



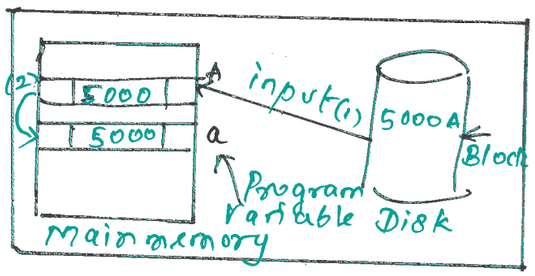
- → App - GUI
- → ODBC - database connectivity
- → Driver Manager - Part of ODBC

TRANSACTION DEFINITION

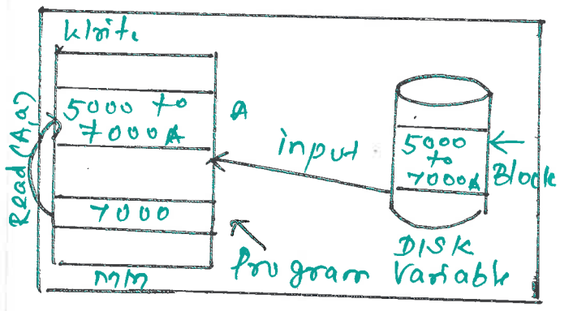
Saving A/c	checking A/c
Rs 5000 to 1000	Rs 1000 to 2000

Transaction
1. Subtract Rs 1000 from Saving (machine crashes)
2. Add Rs 1000 to checking (money disappears)

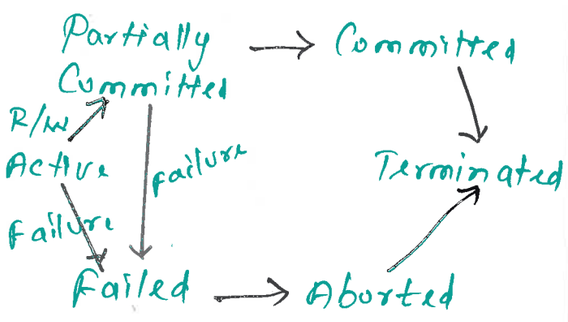
READ



WRITE



TRANSACTION STATES



TRANSACTION PROPERTIES

ACID PROPERTIES
- ATOMICITY (All or Nothing)
- CONSISTENCY Guaranteed
- ISOLATION Committed state independent
- DURABILITY Never Lost

NEED FOR CONCURRENCY CONTROL

- → DIRTY READ PROBLEM
- → UNREPEATABLE READ PROBLEM
- → LOST UPDATE PROBLEM
- → PHANTOM READ PROBLEM

CAUSES OF TRANSACTION FAILURE

- → Computer Crash/failure
- → Transaction failure
- → Exceptions
- → Physical damange
- → Disk failure
- → Concurrency control enforcement

SCHEDULE TYPES

- Schedule
  - Serial
  - Non Serial
- Non-Serial
  - Serializable
  - Non Serializable
- Serializable
  - Conflict
  - View
- Non Serializable
  - Recoverable
  - Non recoverable
- Recoverable
  - Cascading
  - Cascadeless
  - Strict

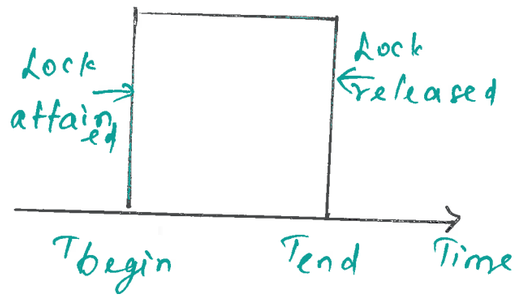
TYPES OF LOCKS

- → Binary lock
  - → locked (1)
  - → Unlocked (0)
- → Shared/Exclusive lock

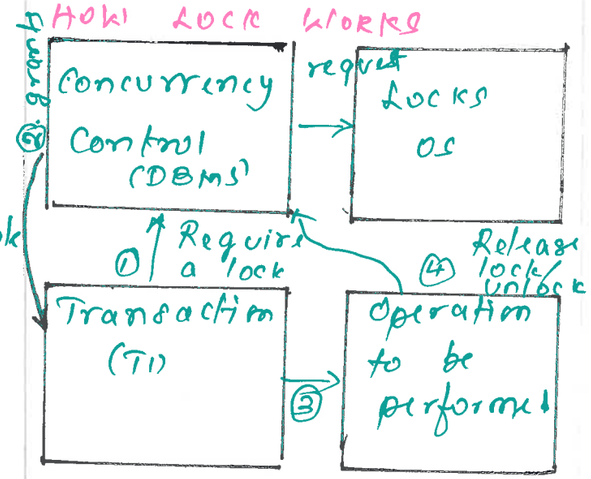
LOCK COMPATIBILITY GRAPH

	S	X
S	T	F
X	F	F

LOCK BASED PROTOCOLS



HOW LOCK WORKS



PITFALLS

- → deadlock
- → Starvation

EXAMPLE

```
LOCK_S(A)
READ(A)
UNLOCK(A)
LOCK_S(B)
READ(B)
UNLOCK(B)
DISPLAY(A+B);
```

Two phase locking (2PL)

- ensure conflict serializable

Phase 1: GROWING

- obtain locks.

Phase 2: SHRINKING

- release locks.

2PL LOCKING CONVERSIONS

Phase 1

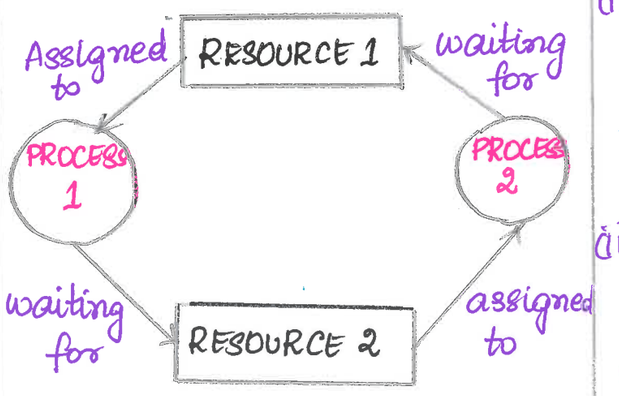
- acquire lock-S
- acquire lock-X
- convert lock-S to lock-X

Phase 2

- release lock-S
- release lock-X
- convert lock-X to lock-S

DEADLOCK:

* T_1 waiting for T_{i-1}



- wait for lock.

DEADLOCK DETECTION

* Build wait-for graph
* cycle found - rollback

DEADLOCK RECOVERY

* Roll back → break deadlock

HOW ROLLBACK?

TOTAL ROLLBACK

* abort & restart fully

PARTIAL ROLLBACK

* starvation
* cost factors.

DEADLOCK HANDLING

- Predeclaration
- Partial Ordering
- Time out based schema
- Timestamp

DEADLOCK PREVENTION

- resource ordering

(i) **WAIT-DIE SCHEME**

- non preemptive
- old transaction wait for young transaction.

(ii) **WOUND-WAIT SCHEME**

- pre emptive
- old transaction wounds
  * force rollback

TIME-STAMP BASED CONCURRENCY CONTROL

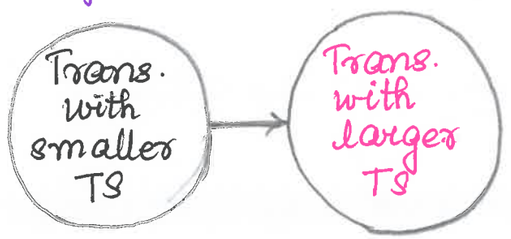
$TS(T_i) < TS(T_j)$

- $TS(T_i)$: timestamp
- old transaction
- T_j: new transaction

* determine serializability

T_i issues write (Q)

- if, $TS(T_i) < R\text{-timestamp}(Q)$
  // Reject write, rollback T_i
- if $TS(T_i) < W\text{-timestamp}(Q)$
  // Reject write, rollback T_i



RECOVERY TECHNIQUES

* Restore DB
* Information in system log.

RECOVERY STRATEGIES

(i) Restore Backup
  - extensive damage

(ii) Identify inconsistency changes
  - non catastrophic failure.

(iii) Deferred update
  - no physical update until commit

(iv) Immediate update
  - update before commit

(v) Undo & Redo
  - multiple times execution

(vi) Caching (buffering)
  - memory buffers
  - cache directory used.

(vii) other strategies
  - checkpoints
  - cascading rollback
  - UNDO/REDO type log entries

SHADOW PAGING

* shadow copy of data
* 2 different places in disk

X', Y' - current copies

X, Y - shadow copies.

- 2 directories.
- No log for single user
- disks: n-fixed size
- new copy of modified page created

* Failure recovery
  - NO-UNDO / NO-REDO
  - discard current directory