DATABASE MANAGEMENT SYSTEM →

∴ DBMS is a collection of interrelated data & a set of programs for convinient and efficient access and storage of data.
(MS) (DB)

- Files → Only D.S. to work in secondary storage

FILE PROCESSING SYSTEM → Conventional, using high level languages. eg:— COBOL

Difficulty → ① Data Redundancy & Inconsistency
(Same data occuring multiple # times) (Having own copy of data in its entry) → WASTAGE OF SPACE
✓ All copies must be updated together
✓ FAILURE → Inconsistency

② Difficulty in accessing data → As most changed requirements for query, new programs have to be written. → TEDIOUS & TIME-CONSUMING

③ Isolation of data → Dif data in dif files & dif formats → gather together in super-list → Difficult but easy to process small files...

④ Concurrency Control → Simultaneous access to same data by multiple processes, mutual exclusion → If not controlled, improper updates

⑤ Serialization → Format of data stored in file

⑥ Security → Who can see/update which data?

⑦ Integrity Enforcement → Assert/Satisfy (in multi-user DBMS environment) data constraints, eg:— Mark = int (≥0 && ≤100), Unique Roll, etc!

(*) All these can be done in High Lvl. Lang. but responsibility of Programmer

(#) DATA ABSTRACTION → Hide complexity of data representation

~~PHYSICAL~~ VIEW 1 ... VIEW 'n' → Part of the data that is accessible for this user

LOGICAL LEVEL — what data is stored? (eg:— Person (roll, name) int, String)

PHYSICAL LEVEL → How data is stored (DSA)

eg:— int a = 10;
4 bytes, 0x10
Structure

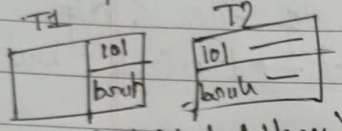
(#) DATA INDEPENDANCE → Change data at one lvl. without affecting the other lvls. eg:— Change PHYSICAL (how) lvl. for faster query → doesnot affect Logical Level (what) → Physical Level Independance

Not Easy
eg:— LOGICAL DATA INDEPENDANCE → Change Logical def^n → App^t Programs still OK

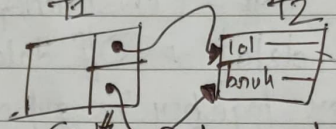
- DATABASE SCHEMA → Overall Structure of Database
  eg:— Student { roll: int, name: string }
- DATABASE INSTANCE → Contents of dB at a point of time
  (temporal) → Changes more frequently
- DATABASE MODEL → Underlying dB, there is a model which show representation of data, their interrelations, constraints, etc.

(i) Record Based Model →

IMP
Relational Model
As a part of record, we store the value of an suitable attribute to maintain relation/association...

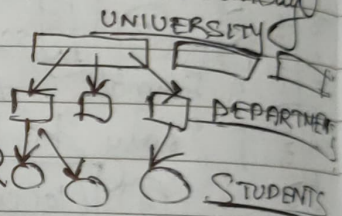


Network Model
Instead of keeping attribute value pointer to related record is kept...



(Record Number maybe)
(or Block Address & Offset in Actual Disk)
Adv → No need of search in T2
Disadv → Deletion & Relocation (need to back-update)

Hierarchical Model
Organisation needed when large # tables arranged according to some hierarchy



(#) LANGUAGE → (1) DDL (Data Definition Language) → Used to specify/modify schema of dB, Easy Statements to create the schema
(2) DML (Data Manipulation Language) → Query/Update/Delete actual data/records of dB
→ Procedural: — Specify what data & how to work (TOTAL PROCESS)
→ Non-Procedural: — Specify what to do (How is taken care of by DBMS)
We work with this in DBMS

FUNCTIONAL COMPONENTS OF DBMS →
(1) Database Manager → Core Software Module (Bit User & OS), Interact with file manager of OS, Concurrency Control, Security, Integrity, Backup & Recovery
(2) Query Processor → Use Non-Procedural DML, Internal Procedural Mechanism is used to reply to the query (efficient detailed strategy & execution plan)

(3) DDL Compiler → Translates DDL statements into metadata (intermediate data about data), which is stored into Data Dictionary. (Who created, Description & Attributes, Permissions, etc...)

(4) DML Pre-Compiler → DML Statements for easy interaction with dB, DML is non-procedural, however in applications, procedural features are needed (eg:- Control Structure, Loop, Variables)

→ Write Application in High Lvl. Lang (Host Lang)

→ In it, use DML to interact with dB (Embedded DML / SQL)

Host language doesn't understand DML

→ DML PRE-COMPILER converts DML into equivalent host language procedural code...

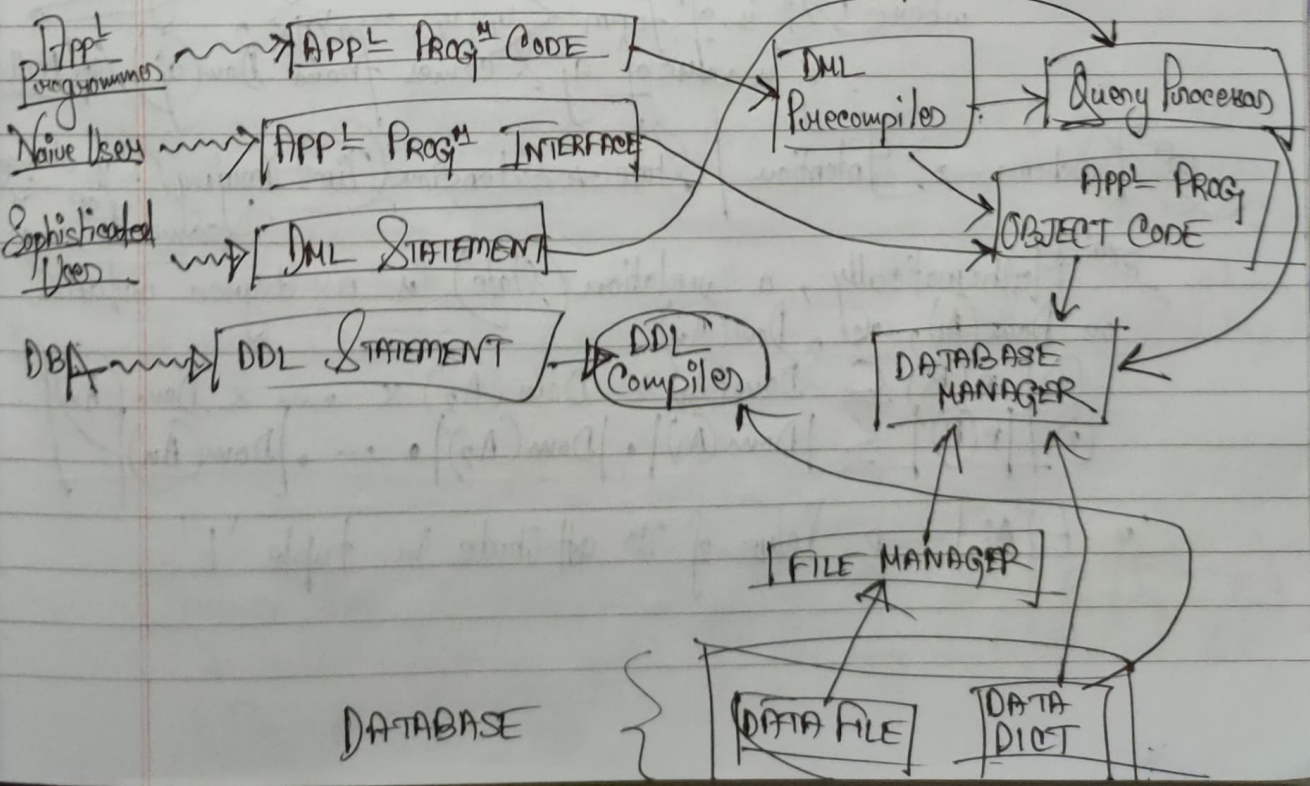
USERS OF DBMS →

(1) DBA (Database Administrator) → Specify / Modify overall schema of dB storage, to grant / revoke permissions / access, usually uses DDL, very less DML required...

(2) APPLICATION PROGRAMMERS → High Lvl. Lang, Embedded DML

(3) SOPHISTICATED USERS → Can directly use DML statements to get results

(4) NAIVE USERS → Use applications interfaces (No HLL, DML, DDL)



9/2/23

RELATIONAL MODEL →

- Relational database is a collection of relations.
- Relation is a set of values / records (table / file of records)

Roll	Name	Score
—	—	—
—	—	—
—	—	—

Roll	CourseCode
—	—
—	—

- Each row of table → Tuple
- Row describes an entity / relationship
- Attribute → Column Header } They help to interpret the data
- Relation Name → Table Name }
- Domain of Attribute → A set of atomic values (cannot be divided further) (Application Dependent)

Every attribute has a domain & it can take values from that domain. Normally, data type & format can be used to specify the domain.

(eg:- In our appⁿ, Roll No is not atomic: 002010501068, val in CSE, Serial No, PG/...)

↳ (D/f attributes can have same domain...)

- Relation Schema → (Overall Structure of Relation) → (aka Intention of a Relation)

$R(A_1, A_2, \ldots, A_n)$ → R is relⁿ name, A_i are attributes taking values from their domains $Dom(A_i)$.

- Degree of relation is the # attributes in relation.

✓ Name of attribute A_i describes the role played by $Dom(A_i)$ in relⁿ R.

[eg:— Roll (int(3)), Score (int(3)) → D/f Meanings, Same Domⁿ]

- Relation / Relation State / Extension of a Relation → $r(R)$ is the relⁿ state of 'n' degree relⁿ $R(a_1, a_2, \ldots, a_n)$.

$r(R)$ is set of 'n'-tuples $\{t_1, t_2, \ldots, t_m\}$, where t_i is of form $\langle v_1, v_2, \ldots, v_n \rangle$

v_j is value of a_j & comes from $Dom(a_j)$ /or/ Null (absence of any value)

[Schema → Intention | State → Extension (Time Varying) (Can change for same Schema)]

Mathematically, a relation (State) is n-degree relation on $Dom(A_1), \ldots, Dom(A_n)$

① $r(R) \subseteq Dom(A_1) \times Dom(A_2) \times \ldots \times Dom(A_n)$

② $|r(R)| \leq |Dom(A_1)| \cdot |Dom(A_2)| \cdot \ldots \cdot |Dom(A_n)|$

- $t[A_i]$ → Value of i^{th} attribute in tuple 't'

Characteristics of a Relation →

1) No 2 tuples are same $[r(R) = \{t_1, t_2, \ldots, t_m\}]$
2) Tuples are unordered → Rel^n M^d doesnot depend on ordering of tuple...
3) Attributes in a tuple may/maynot be ordered...
   $t_i = \langle v_1, v_2, \ldots, v_n \rangle$ → ORDERED
   $t_i = \{\langle A_1, v_1 \rangle, \ldots, \langle A_n, v_n \rangle\}$ → UNORDERED
   Then, $r(R)$ is a set of ← Set of attr value pairs mappings, where each mapping t_i maps $R \to D$
   where $D = Dom(A_1) \cup Dom(A_2) \cup \ldots \cup Dom(A_n)$
   - If ORDERED, Ordering of attributes in schema is followed. (Generally, ORDERED is followed...)
   - [If t_i does not have Key A_j, means $t_i[A_j] = NULL$]
4) Each attribute is (Single-valued) & atomic
   (atomic → cannot be divided further → Appl^n Dep^n)
   (For a tuple, an attribute can have only 1 value...)
   (eg:— For a student, we have only 1 score)
5) The value comes from Domain of Attribute or may be NULL.

Interpretation → A rel^n is a type declaration about an entity/relationship

Constraints → Data in a relation must satisfy the constraints

- MODEL VL CONSTRAINTS → Comes along with the model (eg:— above char)
- SCHEMA VL CONSTRAINTS → Specified at time of schema def^n
- APPLICATION VL CONSTRAINTS → Semantic Constraints (marks ≥ 0 & ≤ 100)
- DATA DEPENDANCY / FUNCTIONAL DEPENDANCY → Designing the database → Judge the goodness of dB design
- DOMAIN CONSTRAINTS → Each attr has domain (specified in schema)

eg:—
```
CREATE TABLE STUDENT
    ROLL NUMBER(3,0)      → Domain
    NAME VARCHAR(25);
```

PRIMARY KEY CONSTRAINT → Subset of attr^n of R ($S_k \subseteq R$) is called SuperKey of R if for any 2 tuples t_1 & t_2, $t_1[S_k] \neq t_2[S_k]$

A subset of Superkey of which no proper subset is Superkey → is called CANDIDATE KEY.

- There can be multiple candidate keys (eg:— Year & Roll OR Year & Registration No.)
- Candidate Key may consist of multiple attributes.

One of the candidate keys is chosen by designer as PRIMARY KEY.

(#) Alphanumeric is preferred over Alphabetic in PK.
(#) Smaller in size (#) In context of application, which one is used for identification in reality...

ENTITY CONSTRAINT → Primary Key cannot be NULL.
NOT NULL CONSTRAINT → name VARCHAR(100) not null

RdB → Set of Rel^n & Constraints → Schema of RdB is $(R_1, R_2, \ldots, R_n)$ where R_i is schema of individual relations & set of integrity constraints (foreign keys)

State → $\{r(R_1), r(R_2), \ldots, r(R_n)\}$

REFERENTIAL INTEGRITY & FOREIGN KEY →

Dept (Referenced) Student (Referencing)

(PK) DCODE	DNAME
D1	—
D2	—
D3	—

(PK) RollNo	Name	DCODE (FK)
—	—	D1
—	—	D2
—	—	D2

(Value of FK must be presence in PK of corr. Rtable → When ALLOWED)

(multiple attr^n together can be Fk)

Pk be Pri. Key of Referenced Rel^n. A subset of attributes Fk is referencing rel^n is foreign key in it that refers to Pk of Referenced Rel^n.

- Domain(Pk in R'ed R^1) = Dom(Fk in R'ing R^2)
- $\forall$ tuple $t_2 \in r_2(R^2)$, either $t_2(Fk)$ is NULL [or] $\exists\ t_1 \in r_1(R^1)$ s.t. $t_1[Pk] = t_2[Fk]$.

IMPACT OF FK ON DML OPERATIONS →

Operation	Referencing Rel^n	Referenced Rel^n
Tuple Insertion	Value added in Fk must be present in Referenced Rel^n	No Impact

Modifying Tuple	If FK value is changed, new value must be in Referenced Relⁿ	(Not Suggested) PK changed, Old Values in Referencing Relⁿ not allowed/updated cascadingly.
Deleting Tuple	No ~~Action~~ Impact	If PK is referred (present in R'ing Rᴸ), then not allowed → FK NULL not allowed in some models