DBMS(SKS)

2.1.26

- Database → Collection of interrelated data
- DBMS → Database + A set of programs for convenient access of data from database
- Data is stored into disk → only data struc associated with sec. storage in file
  ↓
  why not go for file processing system?

- **Difficulties in File Processing System**

— Data redundancy & inconsistency

- when different systems work on common collection of data, data divided into modules → each module ~~are~~ used by different systems, but common data maybe present → this needs to be kept as copies for each separate system → same data is copied at multiple places. This is called REDUNDANCY
- when a system has some data that is derivable from the data of another system — also REDUNDANCY

→ Problem of redundancy → wastage of space; if same info in multiple places & all copies aren't updated simultaneously → causes mismatch. This is called data inconsistency.

→ DBMS single copy of data to be used by multiple systems ~~simultaneous~~ seamlessly

— Difficulty in accessing data

- programs are written to access data as per predefined requirements → if requirement is changed, maybe new programs need to be written

— Data isolation

- data may spread over multiple files gathering data across the files is complex sometimes
  - suppose student data divided as - personal info in a file, result info in another, dept info in another. Thus, collecting all info ~~in~~ from diff files maybe difficult. In DOS environment, a program can at a time open 5 files to ~~20~~ 17 files (3 files are for the prog. itself so 8 to 20)

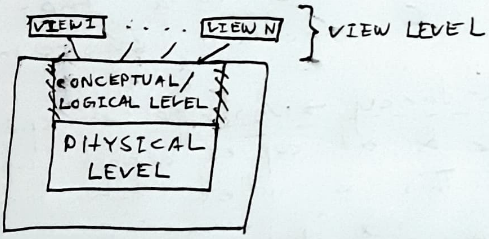
* these issues can be resolved by high-lvl lang. prog. but will depend on skillset of programmer, DBMS provides ~~better~~ inherent handling

— Concurrent Access
- same piece of data is being accessed by no. of processes simultaneously → if not handled properly, various anomalies may occur
→ need for concurrency handled by DBMS

— Security
- all data may not be accessible by every user. Similarly, who can do what & on which data [ACCESS CONTROL]

— Integrity enforcement
- certain properties /constraints of data must be ensured, otherwise, database in invalid state → in DBMS, the system enforces these, user need only mention them, system also determines when to verify the conditions ~~during~~ → DBMS provides set of progs & interfaces for this verification

— Data Abstraction
- hiding complexity of data representation from the user
- In DBMS, there are 3 levels,



- Physical Level ⇒ Lowest level of representation → how the data is stored

- Logical/Conceptual Level ⇒ what data is stored, not how it will be represented

- View Level ⇒ maybe multiple views, part of a database maybe available to a system, not the entire database. Part of db accessible to a user forms their view of db

— Data Independence
- Physical Level Independence → if change in the physical level defn doesn't affect application program
⇒ physical level independence is achievable
e.g. data was in int variable but converted to float
→ if sum variable type is int then not independent but if sum variable type is double then this change won't have any impact → phys. lvl indep.

- Conceptual Lvl Independence → if changes in logical defn don't ~~a~~ lead to change in application prog. then logical lvl independence is achieved

* Conceptual lvl independence is difficult but physical lvl independence is achievable in ~~high~~ -lvl ~~language~~ DBMS

e.g. if DOB is stored & age is stored, then ~~age~~ redundancy present, so if change occurs in ~~any~~ age data, won't affect result since DOB can be used instead → logical lvl independence

- Data Model
  - there exists a model behind the struc of a ~~dat~~ db
  - a collection of tools to describe data interrelation among data, data semantics & constraints
  - → object-based model [ENTITY-RELATION MODEL] → defines relations among data
  - → record-based model → idea of implementation (at conceptual level)
    - → relational model
    - → network model
    - → hierarchical model

○ Relational Model

Database → a collection of relations

STUDENT

Roll (unique)	Name	DOB	MAIL	PH. NO.	DCODE

DEPT

DCODE (unique)	DNAME	YR OF EST.

The 2 record types can be ~~retex~~ related (relation established) by keeping the unique DCODE of dept record as part of student record

- To establish relation b/w diff types of records by keeping certain attribute value as a part of a record

○ Network model (TO KEEP RELN. AMONG THE RECORDS, AS A PART OF RECORD INSTEAD OF ATTR. VALUE, PTR TO RELATED RECORD IS KEPT)

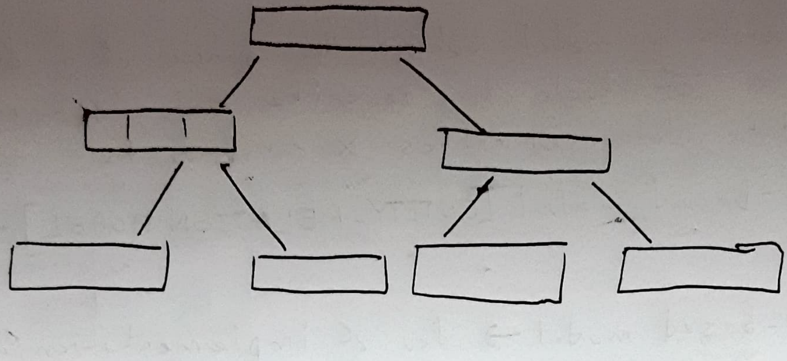
- instead of value of attribute, a pointer to the related record is kept as part of the record

Adv. → pointer access faster than searching by DCODE

Disadv → if record relocated then relation lost

- a forest maybe created through the interrelations b/w several types of records

- **Hierarchical Model**
  - essentially network model but records are arranged acc. to hierarchy (disadv of network model still applies here)
  - instead of forest, the records will be arranged acc. to hierarchy



- **Database Schema & Instance**

SCHEMA – specifies the overall struc of the db; may change but infrequently → static in nature

e.g. Student db → Roll, Name, DOB . . . .

$$\underbrace{\begin{matrix} \text{Roll} & \text{Name} & \text{DOB} \\ \downarrow & \downarrow & \downarrow \\ \text{INT} & \text{STRING} & \text{DATE} \end{matrix}}_{\text{struc. of db} \rightarrow \text{schema of db}}$$

INSTANCE ⇒ content of db at a pt of time is the instance of the db at that timestamp
– it changes frequently i.e. Dynamic → changes with time

– if schema is changed, instance changes too

- **Language to interact with Database**
  - DDL (Data Defn Language) → used to specify the schema of the database i.e. mention the constraints on the data which will be enforced by system
  - DML (Data Manipulation Language) → to add data, manipulate data, delete data, access data
    Can be procedural i.e. how the task has to be done not only what we want to do needs to be specified
    or
    non-procedural – what to be done specified, not how

#Commercial work with SQL (struc. Query Lang.) → contains both DDL & DML statements. The DML statements are non-procedural

- **Functional Units of DBMS**

- Database Manager : it is a software module. It is a core/major part of DBMS
  - Interaction with OS
    - Db stored in disk in form of files. File manager (part of OS) maintains those. ~~Db manage~~
    - Db manager interacts with db to store/retrieve data through file manager
  - Concurrency control
    - automatically done by db manager. If processes go for reading data simultaneously, no constraints
    - if multiple processes go for writing on ~~file~~ db, then constraints to be put → we overwrite automatic concurrency control & write own logic
  - Backup & Recovery
    - Failures may happen → db has to be placed into proper shape (valid state)
    - db manager has to handle - if some processing occurred, it needs to be properly reflected in current version of db, but writing to disk after every process will lead to higher time complexity, if some process couldn't be completed, then old state of db should be retained
  - Integrity Enforcement
  - Security Enforcement

* Thus, except redundancy, every other issue addressed by db manager

- **Query Processor**

translates user given query into a form that DBMS will understand & also convert the same into an equivalent but efficient form. Finally makes detailed execution plan.

- **DML Precompiler**

- DML is non-procedural in SQL but in our appln it may require procedural features like variables, consts, control strucs (if-then-else, loop)
- DML provides efficient way to access db
- Appln. can be developed using both DML & high-lvl lang.

⇒) Appln. is developed using high-lvl lang (we need procedural features) ~~8~~ & in that, the DML statements are embedded (~~ie~~ e.g. ~~SQL~~ DML statement within C prog.)

- However, high-lvl lang (host lang) compiler can't compile embedded DML, we need host-specific DML Precompiler that converts DML statements into equivalent host lang. → now host lang compiler can take care

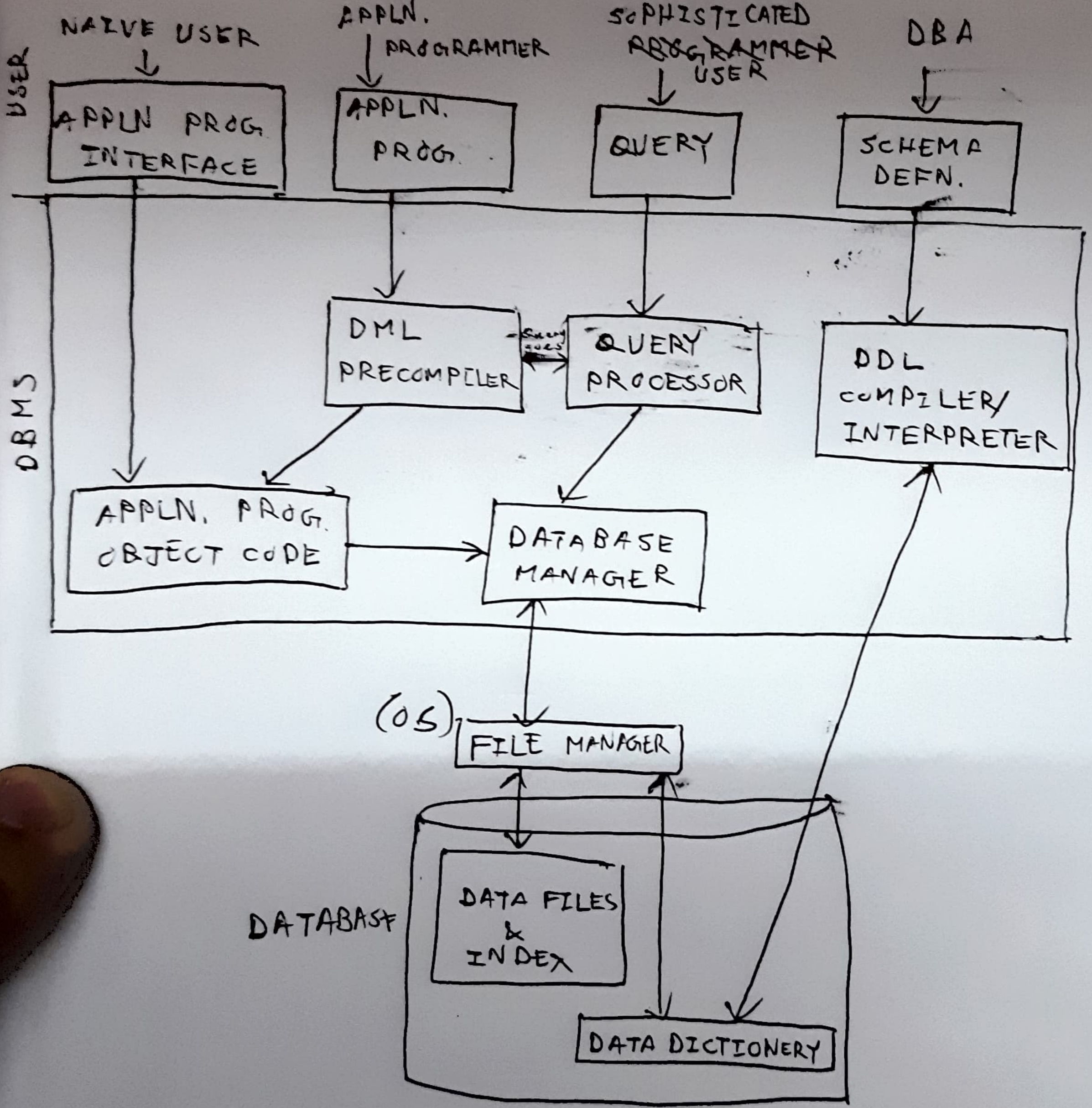
- **<u>DDL interpreter / compiler</u>**
  - in order to handle schema, we write DDL statements
  - DDL compiler translates DDL statements & generates metadata (data about data) which is stored into data dictionary (part of db)

⇒) STUDENT(ROLL, NAME ...)

- Data dictionary has lots of tables, in some tables, it stores the relations, other tables ~~with~~ store constraints on data, creator of data, access rights etc.

- whenever record access is requested, data dictionary is consulted to check all info to determine if access can be granted

- **<u>User (human being)</u>**
  - <u>DBA (Data Base Administrator)</u> — role: to specify / modify the overall schema defn; to specify / modify the storage defn & access mechanism; granting / revoking permission to & from user; high-level integrity specification
    - to accomplish these tasks, DBA has to rely on DDL statements
  - <u>Application Programmers</u> — develop appln programs using high-lvl lang & embedded dml statements; sophisticated users, dml queries; naive users work through appln. interfaces



OVERALL STRUCTURE OF DBMS