DBMS

Data models.

→ files
→ hierarchichal
→ network
→ graph
→ Relational model → collection of tables (structured)
→ Object Oriented
→ Semi-structured / XML

example: MySQL, Oracle, MongoDB, SQL Server.
→ No SQL (graph/semi str.

SQL - structured Query Language

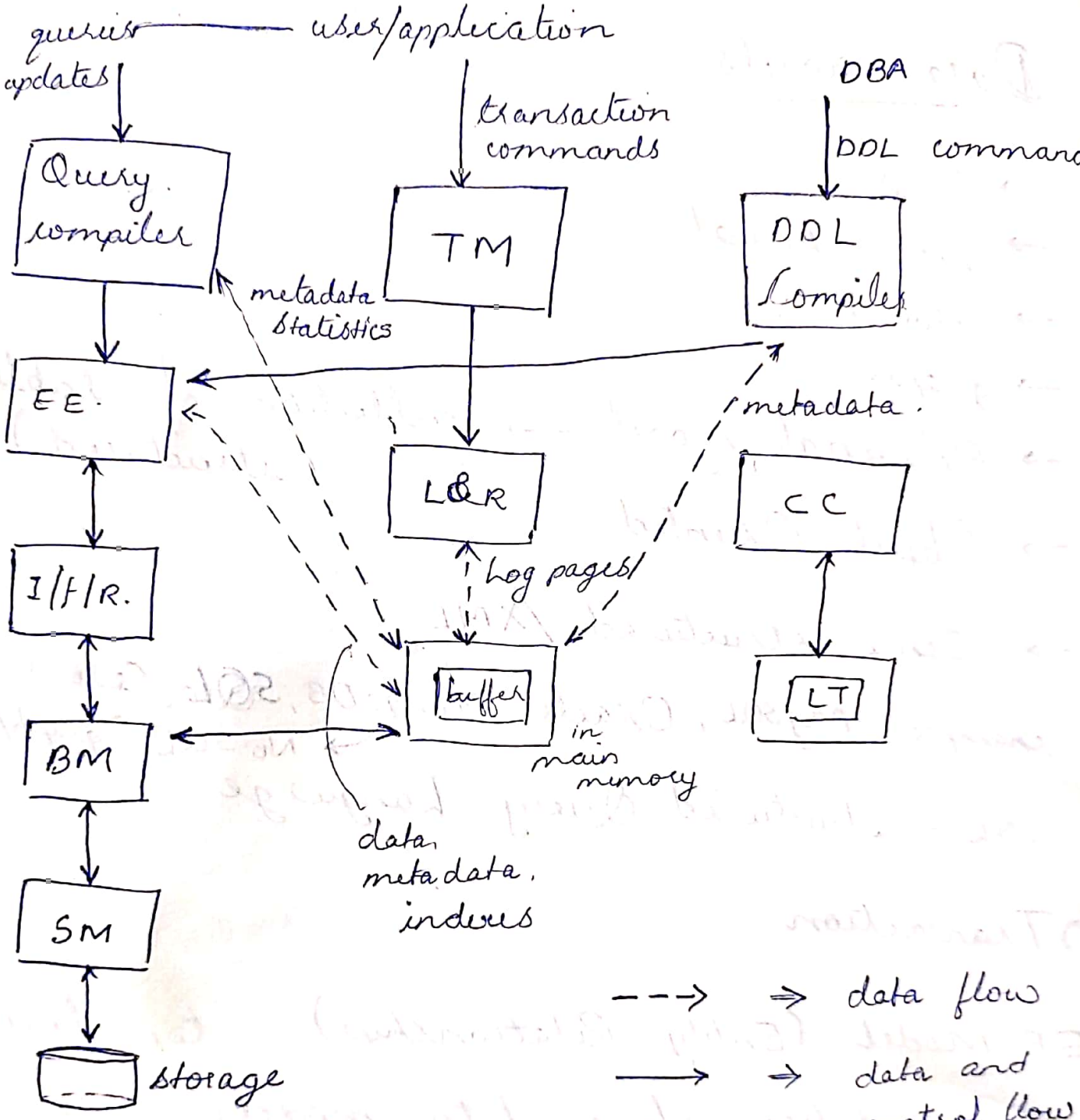
- Transaction

ER Model (Entity Relationship) - top level representation of a data model.

UML (Unified Modeling language)

Data warehousing: Integrating Data bases (heterogenous DB)

Major components of a DBMS



legends.

EE – Execution engine

I/F/R – index, files and record requests

BM – buffer manager

SM – storage manager

TM - transaction manager.

L&R - logging and recovery

DDL - Data definition language.

CC - Concurrency Control

LT - lock table

DML - Data ~~base~~ manipulation lang.

DBA - Data base administrator

User ⟶ application ⟶ DBMS (GUI)

User ⟶ DBMS (CUI)

DBA controls :
- ⊙ Creation of DB
- ⊙ structure of DB (like how many tables are in a DB
- ⊙ Constraints in DB

DBA uses DDL commands for the above function

⇒ User/application interact with DBMS in 2 ways using DML commands

1) queries – retrieve particular set of data (no change in data)

2) update – creation, deletion, update..

⇒ Query compiler converts the DML commands to a query plan that contains operations of relational algebra.

⇒ Query conversion steps : in query compiler
- Query tree is built
- tree is processed
- Query optimiser

⇒ Now, the data which is stored in 2° memory needs to be retrieved to main memory for processing

⇒ indexes are columns that can be used to uniquely identify a row in DB.

⇒ We use B trees and B^+ trees are data structures used to store data in (2°) memory. (They store data only in leaf nodes generally).

⇒ Now, the commands goes through BM and SM (usually in OS) to interact with storage

⇒ we need CC as ~~they~~ these can be multiple transactions in the same DB.

⇒ We need L&R bcoz if a failure occurs in b/w a transaction, the DB must be returned to a safe state using the log files and restart if necessary.

⇒ lock table is used to lock part of DB in case of multiple transaction.

Data model: a notation to describe data or information. It must have

→ structure of data

C uses data structure to handle data which is a conceptual one.

⇒ The physical data model or how data is stored in 2° mem. is different

→ Operations on the data.

— limited functions

→ constraints on data.

1) Relation Model: Structure

eg: Relation - movies

title	year	length	genre
Gone with wind	1939	231	drama
star wars	1977	124	Sci fi
Wayne's world	1992	95	comedy

2) Semi-structured model (XML - extensible modelling lang)

→ hierarchically nested tags (tree like) (like HTML)

→ represent DB as a xml document.

→ elements: contents of documents represented by tags.

→ elements can have sub elements inside and so on.

eg
```
<Movies>
    <Movie title = "Gone with wind">
        <year> 1939 </year>
        <length> 231 </length>
        <genre> drama </genre>
    </Movie>
</Movies>
```

extension of relational model

Object relational Model

Normally, in relational model, values of a column must be atomic (i.e., cannot have a list of values)

But in Object relational model we can have a list of values and operations defined on them.

Terms related to relational model

→ Tables / relation

→ attributes – columns.

→ domain – range of values that an attribute can have.

→ schemas – Name of relation along with its attributes

eg: Movies (title, year, length, genre)

NOTE (order of col. doesn't matter as its a set.

We can have multiple relations in a DB with different schemas

→ Tuples – Values of a row (mention it in the order as given in schema.)

Set
no order.
list
has order of elemn

→ we can incorporate domain also in the schema of a relation

eg: Movies (title: string, year: integer, length: int, genre: string)

→ Equivalent representation: ER or UML diagram

→ Relation instances:

→ schemas change rarely while tuples may change frequently.

DB are generally dynami

Key constraints

Keys of a relation

attribute or a set of attributes of a relation used to uniquely identify the tuples of a relation provided no two tuples are same.

Mention the key in schema also by underlining it

eg Movies (title, year, length, genre)

eg: Movies(

title: string,
year: integer,
length: integer,
genre: string,
studioName: string,
producer#: integer

)

MovieStar(
name: string,
address: string,
gender: char,
birthdate: date
)

Stash(
movietitle: string,
movieyear: integer,
starName: string
)

MovieExec(
name: string,
address: string,
cert#: integer,
net worth: integer,
)

Studio(
name: string,
address: string,
pres C#: integer,
)

If we store all the info in a single DB, some problems like wastage of space, empty fields etc...
So we have design phase.

→ SQL supports both DDL and DML.

Relations in SQL

- stored relation → tables
- views → computational result shown as table (not stored)
- temporary relation → tables generated by sql for the execution of a query

Data types in SQL

1. CHAR(n) - fixed length char string
2. VARCHAR(n) - variable length string of atmost n size

eg: CHAR(5) → |f|o|o|-|-|
i/p. foo
blanks.

VARCHAR(5) |f|o|o| not blanks for rest cells

3. BIT(n) - binary string

BITVARYING(n) - variable size

4. BOOLEAN - T/F

5. INT/INTEGER (SHORT & LONG)

6. FLOAT/REAL (double precision also)

7. DECIMAL(n,d) - decimal no with n digits with decimal point is 'd' from rightmost end

eg: DECIMAL(6,2) ⇒ 0123.45

8. DATE

9. TIME

Defn schema in SQL

```
CREATE TABLE Movies(
        title CHAR(30),
        year INT,
        length INT,
        genre CHAR(10));
```

Defining keys

We can use 'PRIMARY KEY' or 'UNIQUE'

defn primary key using PRIMARY KEY

and candidate keys using UNIQUE

can be set of attribute

eg: (a) title CHAR(30) PRIMARY KEY, or

(b) Inside the definition as

```
CREATE TABLE (
        title CHAR(50)
        year INT
        :
        PRIMARY KEY (title, year)
);
```

(c) or use UNIQUE for candidate keys

The default value of any variable or attribute is NULL if not specified (is the default value)

eg: gender CHAR(1)
DEFAULT '?';

special case { birthday DATE
DEFAULT DATE '00-00-0000' } in definition

Modify relation ⟨ alter
delete

to delete : DROP TABLE table_name;

to alter schema :

ALTER TABLE table_name
ADD new_attribute data_type;

ALTER TABLE table_name
DROP attribute;

Q. Product (maker, model_type)

PC (model, speed, ram, hd, price)

Laptop (model, speed, ram, hd, screen, pri

Printer (model, colour, type, price)

DML using

Relational Algebra

Algebra ⇒ < atomic operands, set of opi

for arithmetic algebra.

operands can be ⌜ variables
⌞ constants

operators ⌜ +, −, ×, /, ...

In relational algebra, variables & op
with relations
↳ core of SQL

Operators – 4

1. Set operations (union, intersection, diffs)
2. removes parts of relation (selection, proj)
3. combine tuples of 2 relation (cross prod, join)
4. Rename operators (Relations, attributes)

) conditions: same no. & type of attributes

) $\Pi_{A_1, A_2}(R)$ ⇒ project attribute A_1 & A_2 from relation R

⇒ to take subset of attribute

$\sigma_C(R)$ ⇒ to take subset of tuples based on some condn C

(returns relation ∴ take care of duplication manually)

$\Pi_{title}\left(\sigma_{length \geq 100 \text{ AND } StudioName = 'Fox'}(Movies)\right)$

④ Relation & attribute name

$\rho_{S(P_1,P_2,P_3)}(R)$ — new relation name & attribute name.

$\rho_S(R)$ — change only relation name

For making operations set compatible

③

R

A	B
1	2
3	4

S

B	C	D
2	5	6
4	7	8
9	10	11

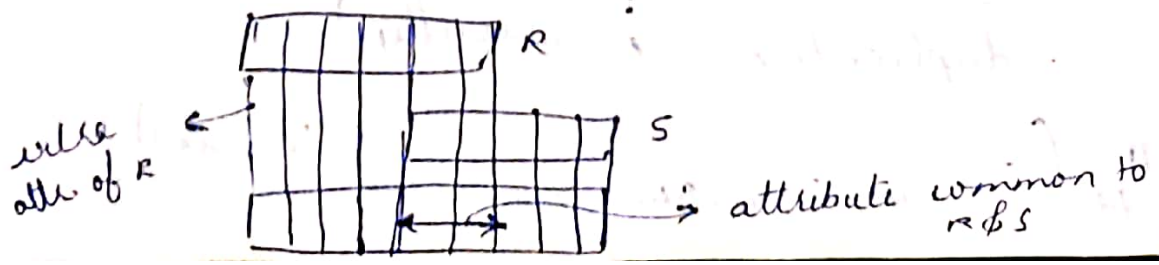
⇒ Cross-product (all combination)

To avoid ambiguity, rename.
(usually. (relation. attr_name))

⇒ joins ↗ if both relation has same attributes which r to be used for join condn

ⓐ Natural join. $(R \bowtie S)$ / $R * S$.

?



Common attributes of 2 relation exist, then only the tuples with equal values for that in R & S, are joined

ii

A	R.B	S.B	C	D
1	2	2	5	6
3	4	4	7	8

$R \bowtie S$

3) Theta join (α_c — condition to be satisfied) → no natural join followed by condn

eg:

A	B	C
1	2	3
6	7	8
9	7	8

(U)

B	C	D
2	3	4
2	3	5
7	8	10

(V)

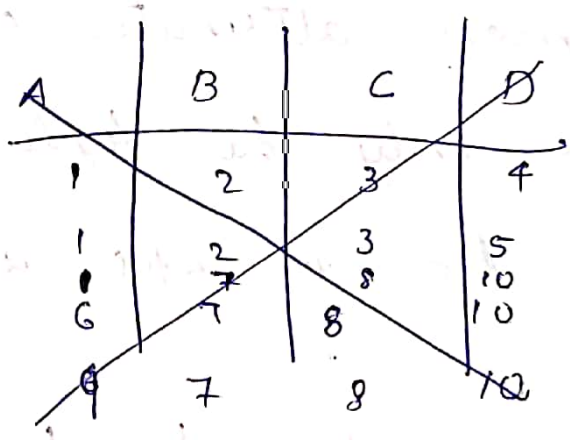
$U \bowtie V \Rightarrow$

A	U.B	U.C	V.B	V.C	D
1	2	3	2	3	4
1	2	3	2	3	5

use only 1 copy of an attribute

A	B	C	D
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

$U \bowtie_{A<D} V \Rightarrow$



since equality is not checked we need B & C of V & U seperately

A	U.B	U.C	V.B	V.C	D.
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

ii $U \bowtie_C V \Rightarrow \sigma_C (R \times S)$

$\Pi_{title}(\sigma_{len \geq 100 \text{ AND } Studio = 'Fox'}(R)) \equiv \Pi_{title}(\sigma_{len \geq 100}(R) \cap \sigma_{studio}$

SQL

```
CREATE TABLE table_name(                          ← default allows NULL
        DNAME       VARCHAR(10)     NOT NULL
        DNUM        INTEGER         NOT NULL.
        MGRSSN      CHAR(9),
        MGRSTRTDATE CHAR(9),
        PRIMARY KEY (DNUM),
        UNIQUE (DNAME),
        FOREIGN KEY (MGRSSN) REFERENCES EMPU   ← table name of referenced relation
        ON DELETE SET DEFAULT ON UPDATE CASCADE);
                                  ↑ foreign key.
```

↙ deletion from referenced relatn.

```
ALTER TABLE table_name ADD JOB
        VARCHAR(10);
```

⇒ If more than one foreign key occurs define they along with ON DELETE & ON UPDATE (no comma after foreign key before ON DELETE)

⇒ A bag / multi-set set with duplicate entries (no necessary).

⇒ SQL query returns bag while relation algebra returns set.

⇒ SELECT <attr_list> FROM <TABLE LIST>

WHERE <condn.>

⇒ If same attribute name is there refer it using relation name.

eg: If u have to get name of manager of each employee

```
SELECT   E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM     EMPLOYEE E, S
WHERE    E.SUPERSSN = S.SSN
```

* Also use 'AS' like FROM EMPLOYEE AS E, EMPLOYEE AS F

Q10. If WHERE clause is not given, its assumed to be WHERE TRUE and for more than 1 relation, cartesian product is displayed.

'*' ⇒ all attributes

'DISTINCT' ⇒ returns SET rather than bag.

SET operations

→ give a set as o/p (not a multiset)

→ usually 'OR' in question may use union

NESTING OF QUERIES

```
Select
From
where ( Select
        From
        where ...
      )
```

eg:
```
Select name
From employee
where DNO (IN) (
    Select DNUM
    from Department
    where Dname = 'Research'
```

IN returns TRUE if a value v is present in a set of values (same attribute)

Correlated nested query

If an attribute of relation in outer query is used in inner query, its said to be correlated.

eg:
```
Select  E.FNAME , E.lastname
FROM    Employee AS E
where   E.SSN IN
        (SELECT   ESSN          → unqualified attribute refers to innermost relation
         FROM     DEPENDENT E   → outer query's
         WHERE    ESSN = E.SSN AND
                  E.FNAME = Dependent_name)
```

Retrieve name of employee having same 1st name as that of his dependent

CONTAINS : used in older SQL's to check 2 sets of values i.e. if elements of 1 set is in the other or not.

If all are present, it returns TRUE

EXISTS : Check whether is result of correlated query exists/is empty or not.

EXPLICIT SETS : check in a particular list

eg: Select distinct SSN
FROM works_on
where PNO IN (1, 2, 3)

NULLS : IS or IS NOT is used to compare NULLS. (each NULL value is diff from other NULL value ∴ no '=' works)

JOINED RELATION :

eg: Select name
From emp, dept
where DNum = DNO AND Dname = 'Research'

or

```
select  name.
FROM    (EMP JOIN dept
            ON DNUM = DNO)
where       Dname = 'Research'
```

or.

```
Selut   name.
FROM    (Emp NATURAL JOIN dept
            AS (DEPTNAME, DNUM, MSSN)
where       Dname = 'Research'
```

Aggregate fn.

→ Count, max, min, SUM, AVG...

```
Select  COUNT(*)
FROM    employee
```

GROUPING

→ Each sub group of tuple contains tuples having same value for an attribute.

eg: SELECT DNO, COUNT(*), AVG(SALARY)
(DNO - grouping attribute)
FROM EMLOYEE
GROUP BY DNO

⇒ o/p: give count & Avg(salary) of employee in same dept

HAVING - CLAUSE

Retrive values of these fn for only those groups that satisfy some condn.

eg: Select PNUM, PNAME, COUNT(*)
FROM PROJ, WORK_ON
where PNUM = PNO
GROUP BY PNUM, PNAME
HAVING COUNT(*) > 2.

SUB STRING COMPARISON

→ 'LIKE' : to compare partial string

eg: selut name
from employee
where ADDRESS LIKE '% NITC %'

% ⇒ any number of char

_ ⇒ one char

ARITHMETIC OPERATION

selut name, 1.1 * salary
from employee.

ORDER BY

→ sort the tuples in a query result based on values from some attribute

→ default ASC

ORDER BY DNAME, DNUM → order by DNAME & among them order by DNUM

INSERT

→ INSERT INTO table_name VALUES (' ', ...);

in the order of attributes in defn.

→ to insert only specific attribute values:-

→ INSERT INTO table_name (attributes) VALUES (' ', ...);

⊙ rest are filled with default values.

→ to insert multiple tuples:

```
INSERT INTO table_name(attribute)
SELECT attribute
FROM table_name(s)
WHERE condn.
GROUP BY attribute
```

here a temporary table is created. Hence changes in base tables won't affect this table.

For that you need to create a

VIEW

DELETE (tuple deletion)

```
DELETE  FROM  table_name
WHERE         condn(s)
```

UPDATE

```
UPDATE  table_name
SET     DNAME = 'Res', DNUM = 1
WHERE   condn.
```

```
SELECT
FROM
[WHERE
[GROUP BY
[HAVING
[ORDER BY
```

} general query structure

Constraints as Assertion

```
CREATE  ASSERTION  const_name
CHECK  (condn to be checked)
```

→ no emp. has sal > manager

eg: (NOT EXISTS (SELECT * FROM EMPE, EMPM, DEPT D
WHERE E.SAL > M.SAL AND
E.DNO = D.NUMBER
D.MGRSSN = M.SSN))

IMP

Triggers

→ Monitor database & take initiate action when a condn occurs.

→ Syntax III" to assertion & has

- event
- condn
- action

VIEWS

→ limited update operations & table may not be physically stored

→ updated as base is updated

- CREATE VIEW view_name AS

```
SELECT attribute(s)
FROM table(s)
WHERE
GROUP BY ;
```

→ Can write complex query - select query. as view & u as to not repeat it again & again

- DROP view_name;

Database programming

→ To access database from app. pgm

Approaches:

→ embedded commands

→ library of DB fns

Embedded SQL

- embed SQL query inside a host pgming lang like C/C++, java...
- EXEC SQL ... $ END-EXEC ~~SQL~~

Start & end of SQL query inside C/C++, java..