Chapter - 1
Introduction

1.1 File Processing System

The information on a system can be stored in permanent files. The system uses a no. of application programs, each of them is designed for data manipulation. As the system needs arises, new applications are added to the system. Such system is known as file processing system.

For eg: consider a traditional library management system that uses file system to store data. The system consists of application programs to allow users to manipulate data. Such programs may be used to issue book, add student records, return book, fine calculation and so on.

1.2 Disadvantages of file processing system

1.2.1 Data Redundancy

File system can lead to data redundancy, a situation in which the datas are to be updated in several files. It results in storage of same information in several files consuming high storage space and cost. Also, the information may not in updated in all files causing inconsistency.

1.2.2 Integrity Problems

Each data stored must satisfy certain consistency constraints which are coded in application programs. In order to enforce new constraints, it is difficult to make changes in the program, instead new program must be added. This unassure the maintenance task.

1.2.3 Concurrency Access

Concurrency means ability to allow multiple users access to same record without affecting the transaction processing. In file system, when an application open a file, that file is locked and no one else can access that file at the same time.

1.2.4 Atomicity Problems

Atomicity means ability to recover exact state in case of system failure. In file system, the system can not ensure the operation is completed or it is not in action resulting in unambiguous state in case of system failure.

1.3 Database Approach

A database is a shared collection of related data or information. It can be viewed as a data repository that is defined once and accessed by various users.

A database has following properties:-

a) It is representation of some aspect of real world.
b) It is logical, coherent and internally consistent.
c) It is designed, built and populated with data for specific purpose.
d) Each data item is stored in a field.
e) The combination of fields forms a table.

Database management system (DBMS) is a collection of programs that enables users to control all access to databases that makes data storage and retrieval efficient.

1.4 File System Vs. DBMS

→ A database is self describing as it also contains metadata that defines the data and relationships between tables.

In file-based system, data definition is embeded with application programs.

→ In file system, structure of data files is defined in programs. So, if changes to file structure is needed, the programs should also be changed.

In DBMS, data structure is stored in system catalogue independent of programs.

→ Multiple users can access the same database at the same time maintaining data integrity.

→ Database control data redundancy as each data item is stored in only one place in database.

→ Data sharing among individuals having access for it is allowed with a database.

→ Database allows restriction of unauthorized access. A user can have access limited by the system.

→ DBMS allows data independence by separating metadata from application programs.

1.5 Data Abstraction and Data Independence

1.5.1 Data Abstraction

Database system is a complex bulk of data structures. Within a system, there may be many irrelevant details for the users. The process of hiding such irrelevant details from users is known as data abstraction.

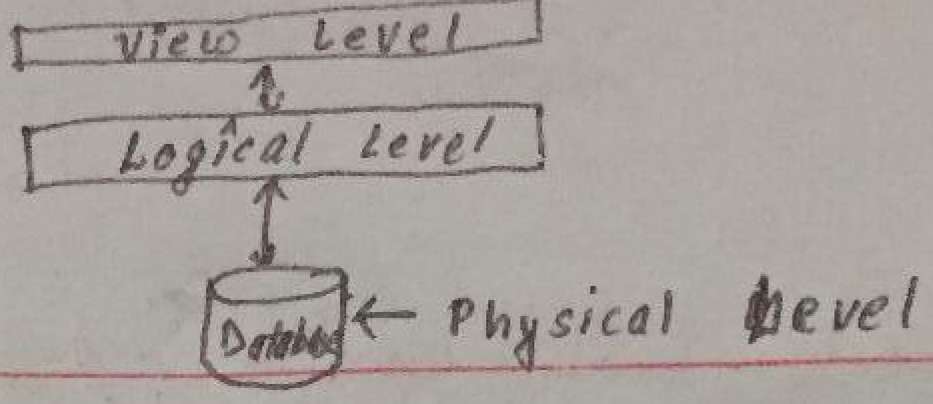
1.5.2 Data Abstraction Levels

→ Physical Level

The physical level is the lowest data abstraction level. It describes the actual method of data storage in database revealing the complex data structure details.

→ Logical Level

Logical level provides higher data abstraction than physical level. It describes the type of data stored in database.



→ View Level

The view level is the highest level of data abstraction. It describes the user interaction with database system.

1.5.3 A case study: Data Abstraction Levels

Let us consider that we are storing customer information in a customer table.

→ At physical level, these records are described as blocks of memory storage space.

→ At logical level, they are described as fields and attributes along with the logical relationships.

→ At view level, user just interact with the help of GUI and enter details at the screen.

1.5.4 Data Independence

A database system contains a lot of data along with user's data. A set of metadata needs to change over time to satisfy user's needs. So, metadata follows a layered architecture, so that change of data in one layer does not affect data at another level. Such property of data is known as data independence.

1.5.5 Types of data independence

→ Logical Data Independence

Logical data represents the data about database such as how data is managed, relationship between tables and so on. Logical data independence is the ability to change logical schema without changing external schema or user views. It implies that the function of application should not be affected on changing logical schema.

For eg: if we changes some table format, it should not change the data residing on the disk.

→ Physical Data Independence

Physical data independence is the ability to change the physical data or model without affecting the logical data. It deals with hiding the details of storage structure from user applications.

For eg: if we want to upgrade the storage system to replace hard-disks with SSD, it should not affect the logical schema.

1.6 Schema and Instances

1.6.1 Schema

A schema is the skeleton structure of database representing its logical view. It defines the entities and relationships among them along with formulation of all the constraints.

1.6.2 Instance

A database instance is a state of operational database at any given time with data within it.

1.7 Concepts of DDL, DML and DCL

1.7.1 Data Definition Language (DDL)

DDL deals with database schemas and descriptions, of how data should reside in the database.

Eg: CREATE → to create database and its objects
ALTER → to alter the structure of existing database
DROP → to delete objects from database

1.7.2 Data Manipulation Language (DML)

DML deals with data manipulation process such as to store, modify, retrieve, delete and update data in database.

Eg: SELECT → retrieve data from database

INSERT → insert data into table

DELETE → delete all records from database table.

1.7.3 Data Control Language (DCL)

DCL deals with rights, permissions and other controls of the database system.

Eg: GRANT → all users access privileges to database

REVOKE → withdraw users access privileges.

Chapter 2
Data Models

2.1 Data Model

A data model is an abstract model that organizes elements of data and depicts how they relate to each other and to the properties of the real world. It explicitly determines the structure of data.

2.2 Perspectives of Data Model

→ Conceptual data model

Conceptual data model describes the semantics of a domain, providing concepts for presenting data in ways that are close to the way people perceive data. It consists of entity classes for a domain and the relationship assertions showing association between pairs of entity classes.

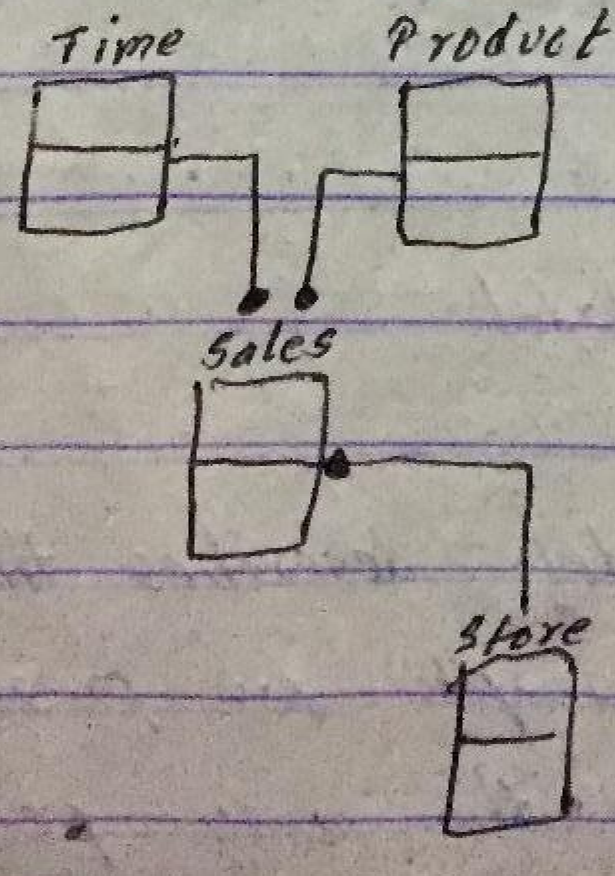
→ Logical data model

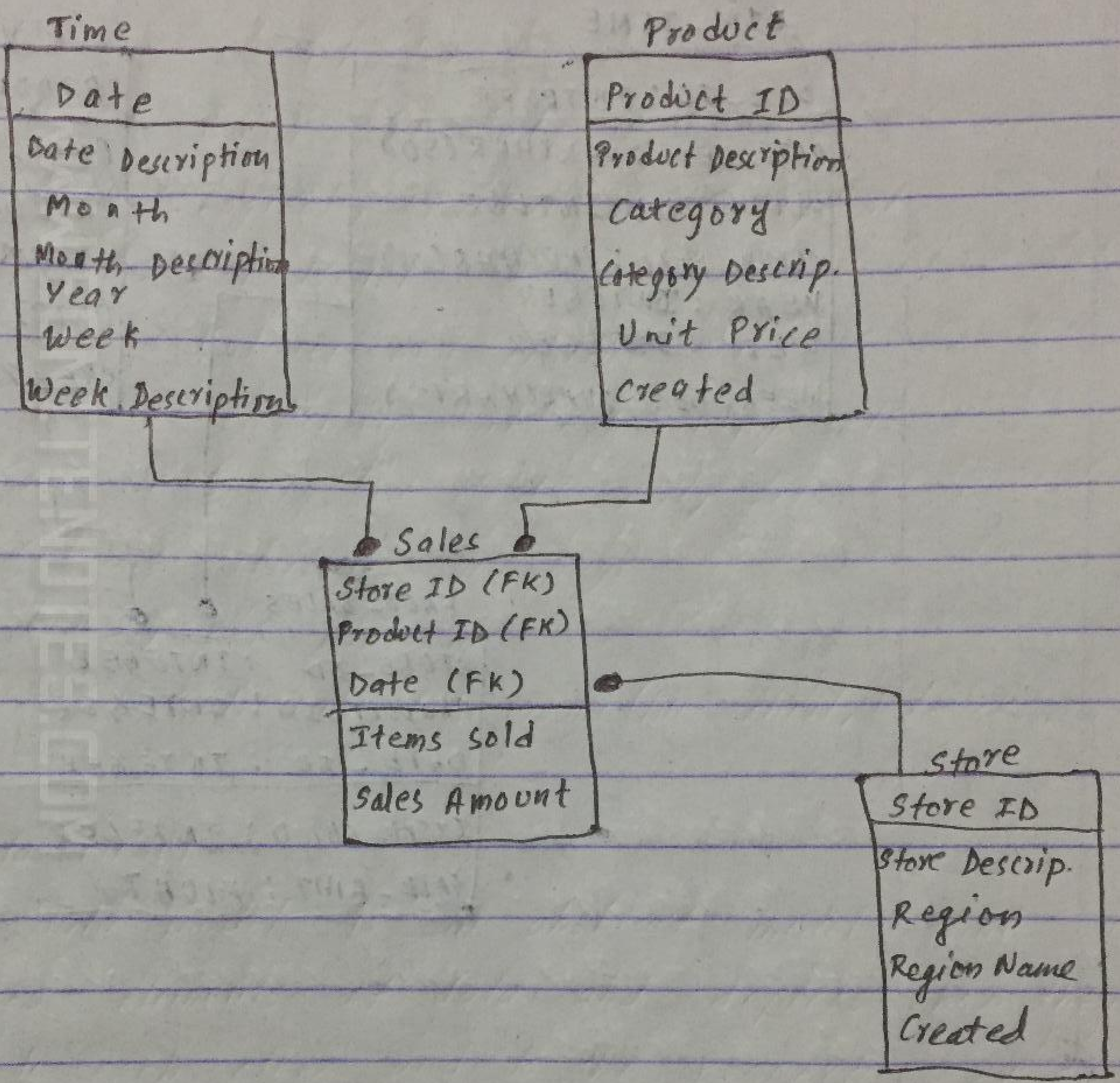
Logical data model describes the semantics, as represented by a particular data manipulation technology. It includes descriptions of tables and columns, object oriented classes, primary and foreign keys.

→ Physical data model

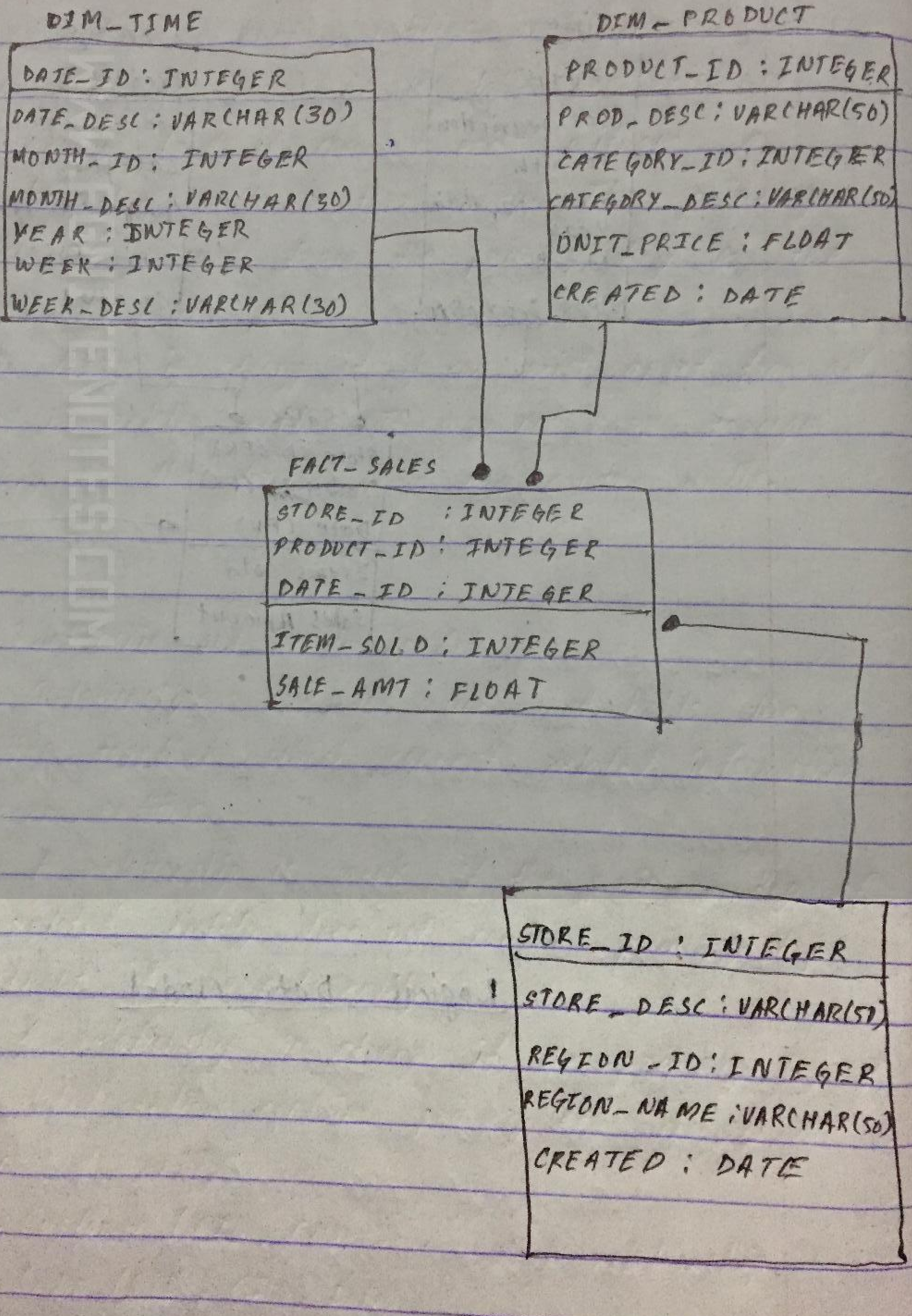
Physical data model describes the physical means by which data are stored. It includes partitions, CPUs, tablespaces and so on.

→ A case study:





Logical Data Model


DIM_TIME
DATE_ID : INTEGER
DATE_DESC : VARCHAR(30)
MONTH_ID : INTEGER
MONTH_DESC : VARCHAR(30)
YEAR : INTEGER
WEEK : INTEGER
WEEK_DESC : VARCHAR(30)
DIM_PRODUCT
PRODUCT_ID : INTEGER
PROD_DESC : VARCHAR(50)
CATEGORY_ID : INTEGER
CATEGORY_DESC : VARCHAR(50)
UNIT_PRICE : FLOAT
CREATED : DATE
FACT_SALES
STORE_ID : INTEGER
PRODUCT_ID : INTEGER
DATE_ID : INTEGER
ITEM_SOLD : INTEGER
SALE_AMT : FLOAT
STORE_ID : INTEGER
STORE_DESC : VARCHAR(50)
REGION_ID : INTEGER
REGION_NAME : VARCHAR(50)
CREATED : DATE

2.3 E-R Model

E-R model is based on the notion of real world entities and relationships among them. It is used generally for conceptual design of database. The E-R model creates entity set, relationship set, general attributes and constraints.

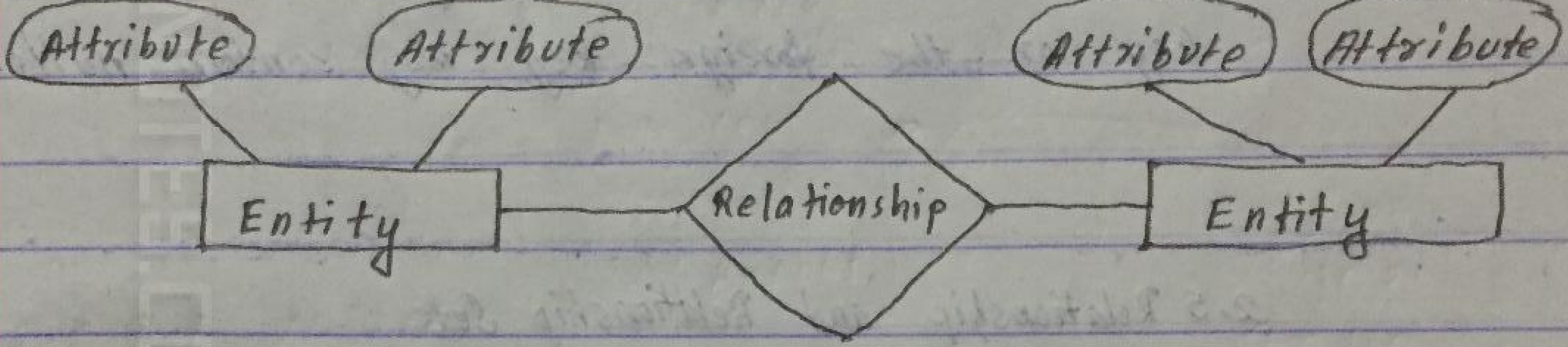


Fig. Schematic of E-R Model

2.4 Entities and Entities sets

An entity is an object in the real world with an independent existence that can be differentiated from other objects. For eg: car, lecturer, course, etc.

An entity set is a collection of an entity type at a particular point of time.

An entity type is a collection of similar entities.

Based on strength, entities are classified as:

→ An entity is weak if its tables are existence dependent. (i.e. it can not exist without a relationship with another entity.) Its primary key is derived from the primary key of the parent entity.

→ An entity is strong if it can exist apart from all of its related entities. It may not have foreign key or the foreign key may contain nulls.

2.5 Relationship and Relationship Sets

Relationships: are the link that holds the entities together which connect related information between entities.

A relationship is weak if the primary key of the related entity does not contain primary key component of parent entity.

A relationship is strong if the primary key of the related entity contains primary key component of parent entity.

Eg: Customer (<u>CID</u>, CName)
Order (<u>OID</u>, CID, Date) } Weak relationship

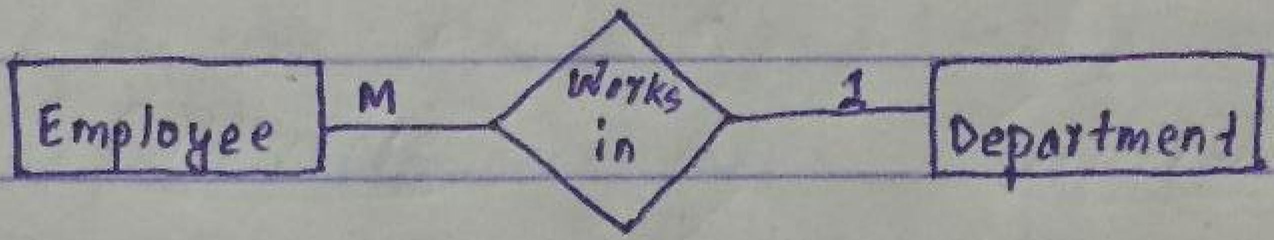
Course (<u>CCode</u>, DCode)
Class (<u>CCode</u>, <u>Section</u>, Time) } Strong relationship

2.5.1 Types of Relationships

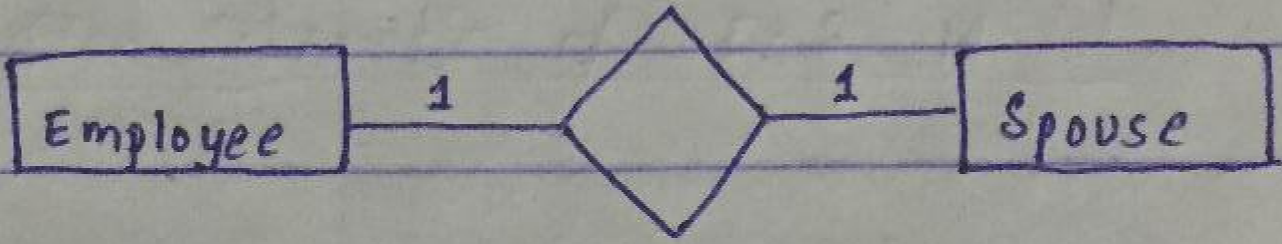
→ One to many (1 : M)

Here, one entity relates to other entity such that single instance of a entity is related to many instances of another entity.

Eg: one department has many employees.



→ One to One (1 : 1)



→ Many to Many (M : N)

→ It can not be directly implemented.

→ It is implemented by breaking up to produce a set of 1 : M relationships

2.6 Attributes and Keys

→ Attributes are the fields that describes the entity.

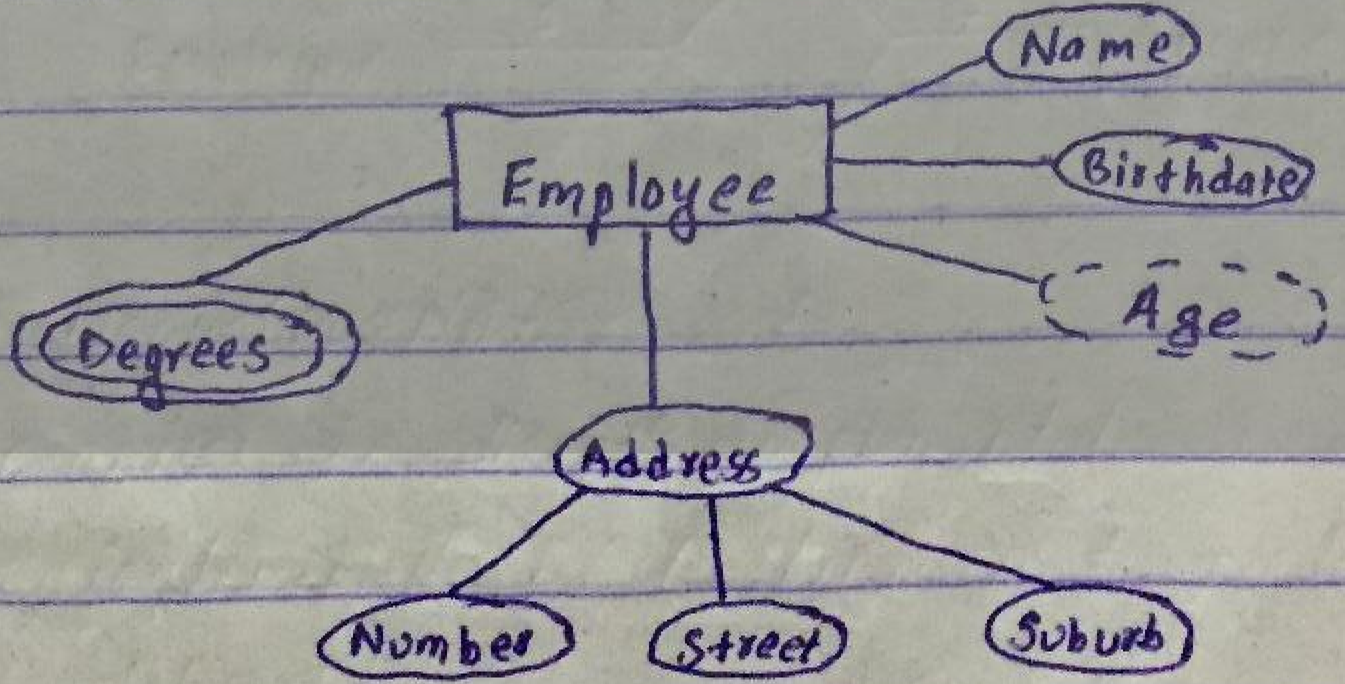
→ Each attribute has a name and is associated with an entity.

→ Simple attributes are those attributes with a single value. Eg : Name = {John}

→ Composite attributes are those that consist of a hierarchy of attributes. Eg : Address = {59 + 'Meek Street' + 'Kingsford'}

→ Multivalued attributes are those having a set of values for each entity.

→ Derived attributes are those with values calculated from other attributes.

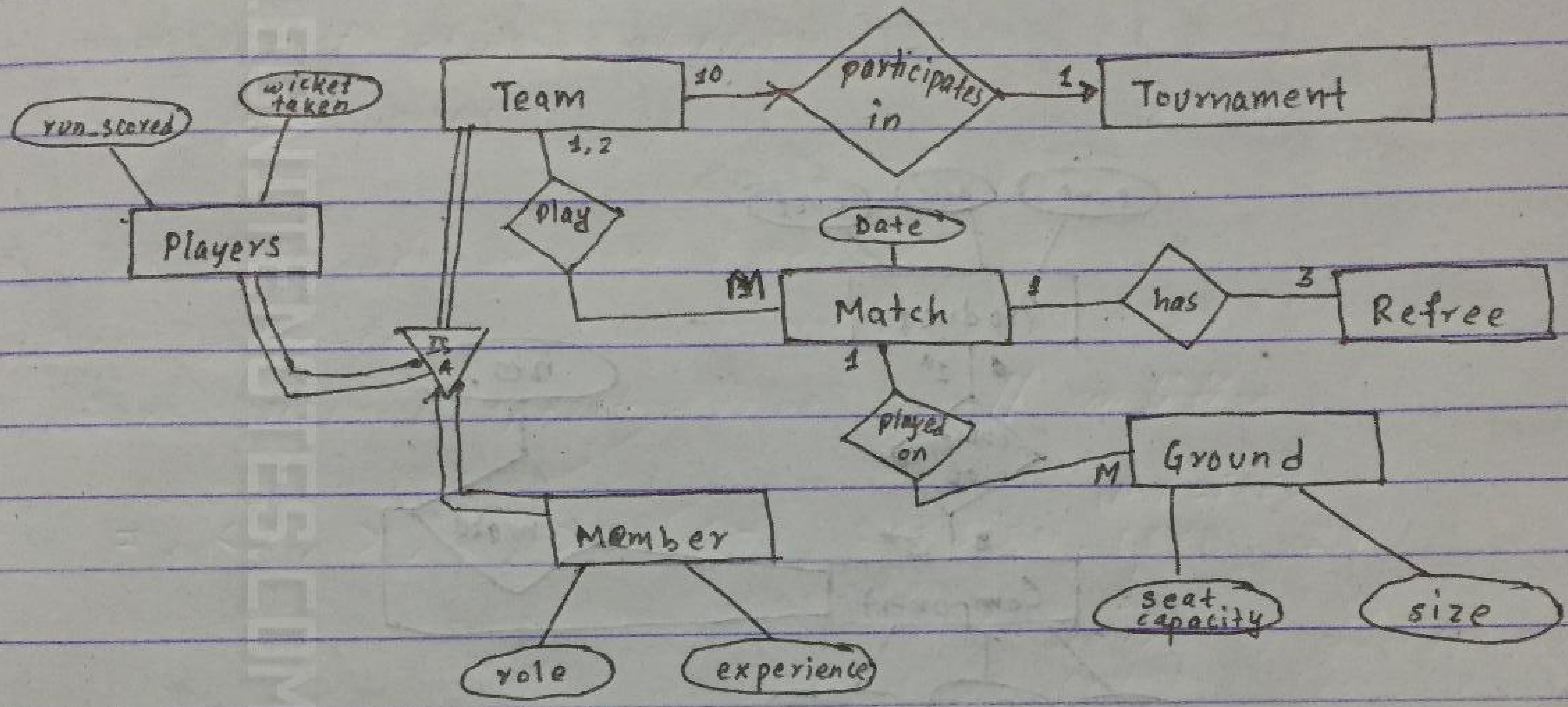


→ Key is an attribute or a group of attributes whose values can be used to uniquely identify an individual entity in an entity set.

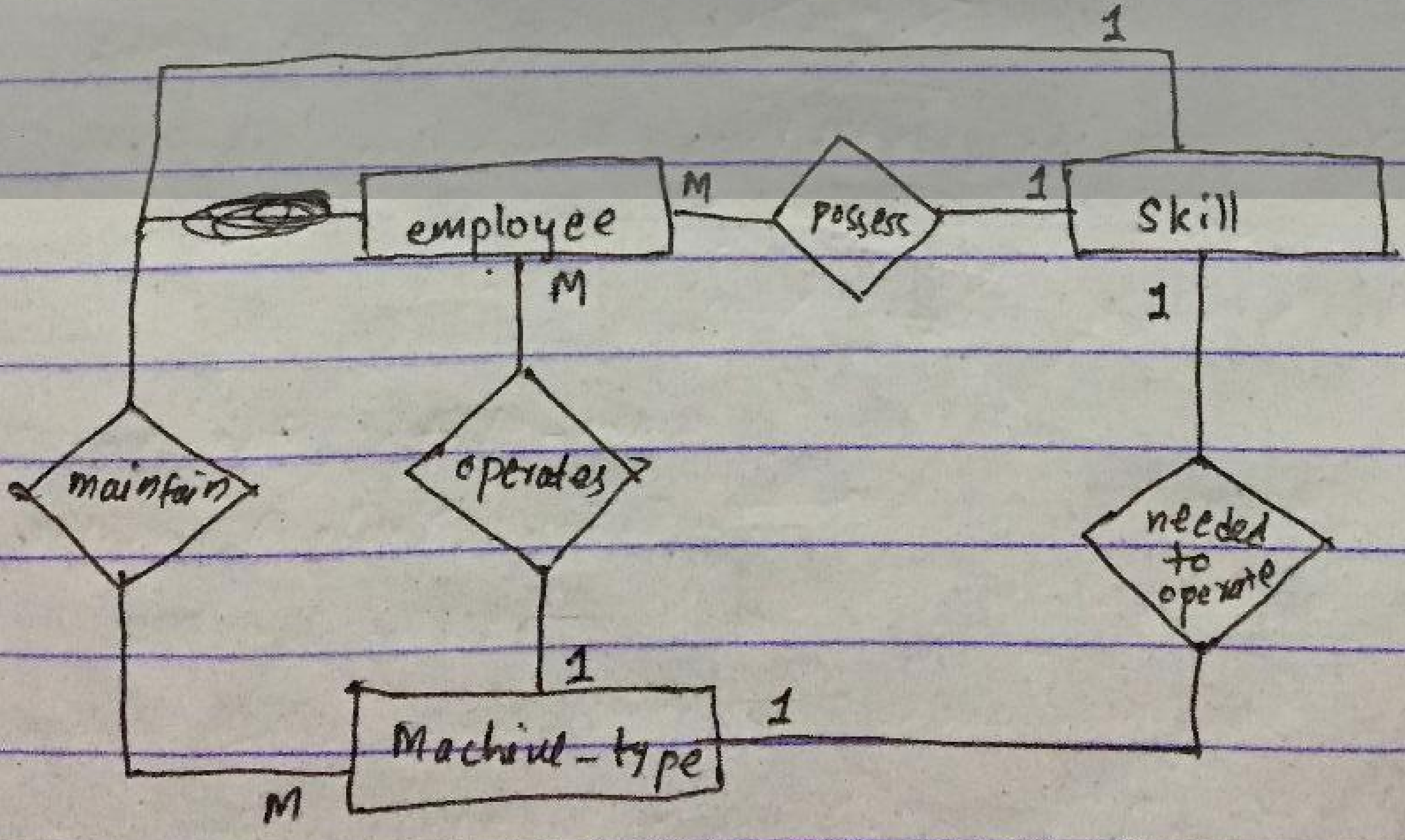
Chapter-2

ER Diagram

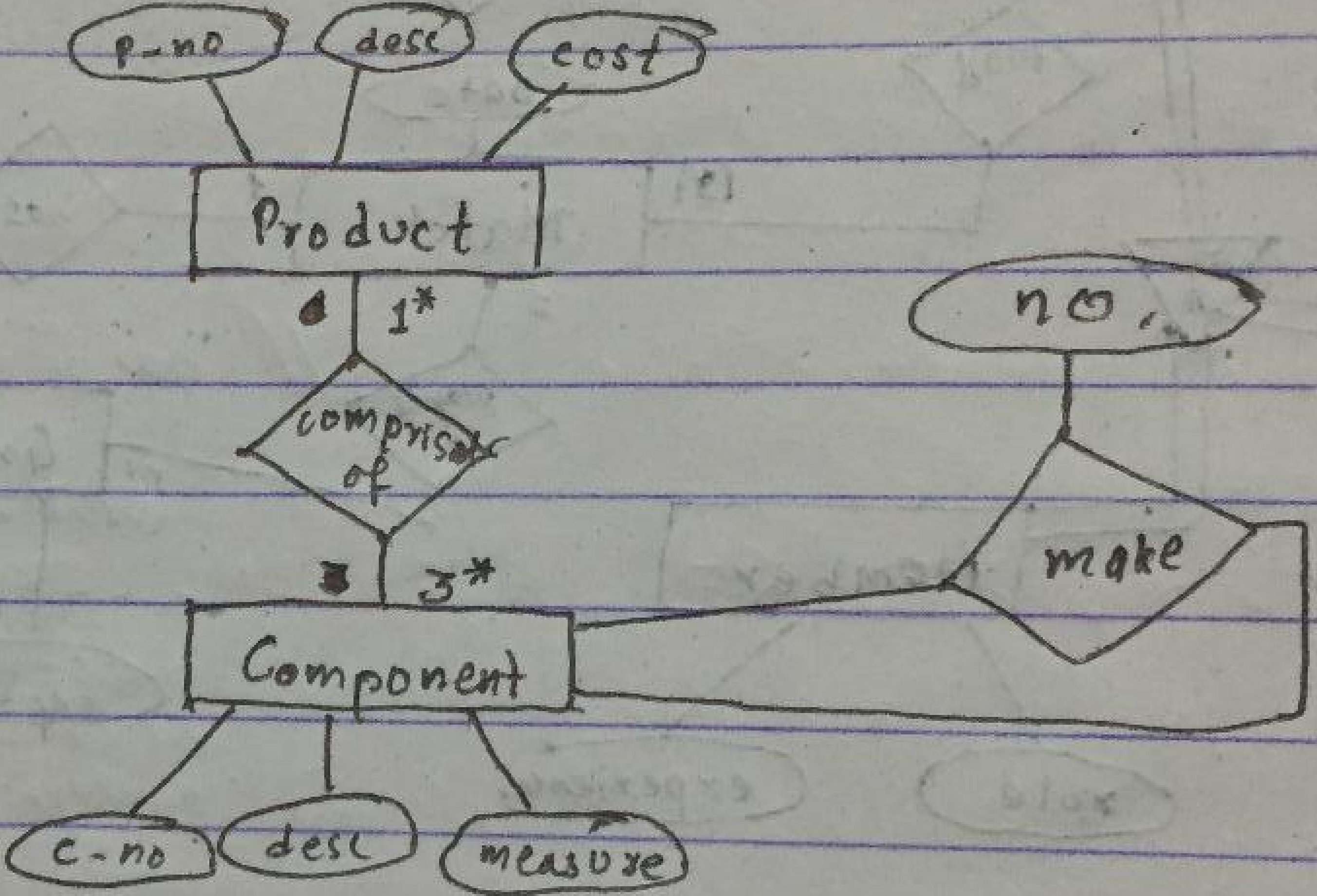
#2072 Magh



2070 Bhadra

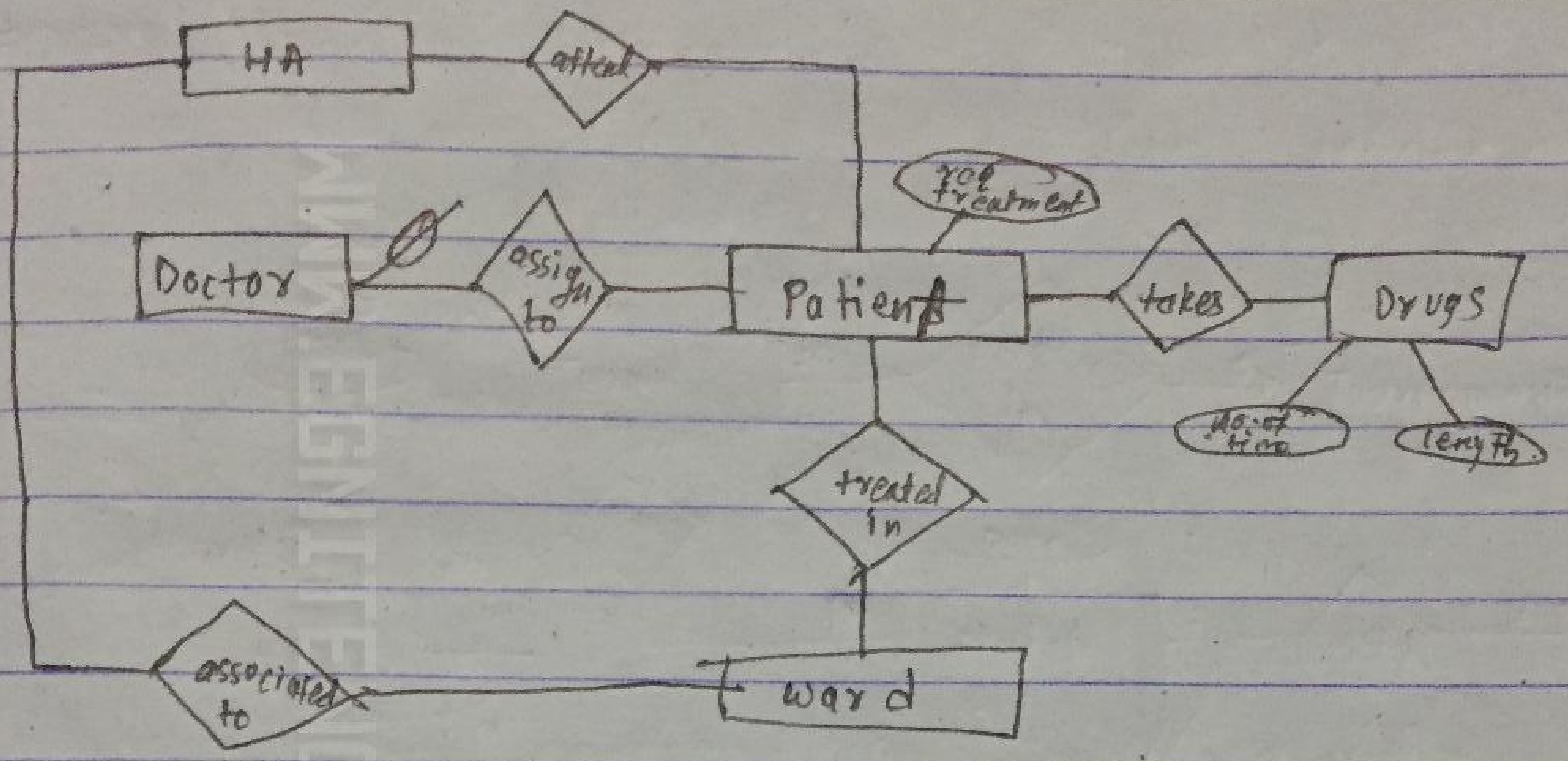


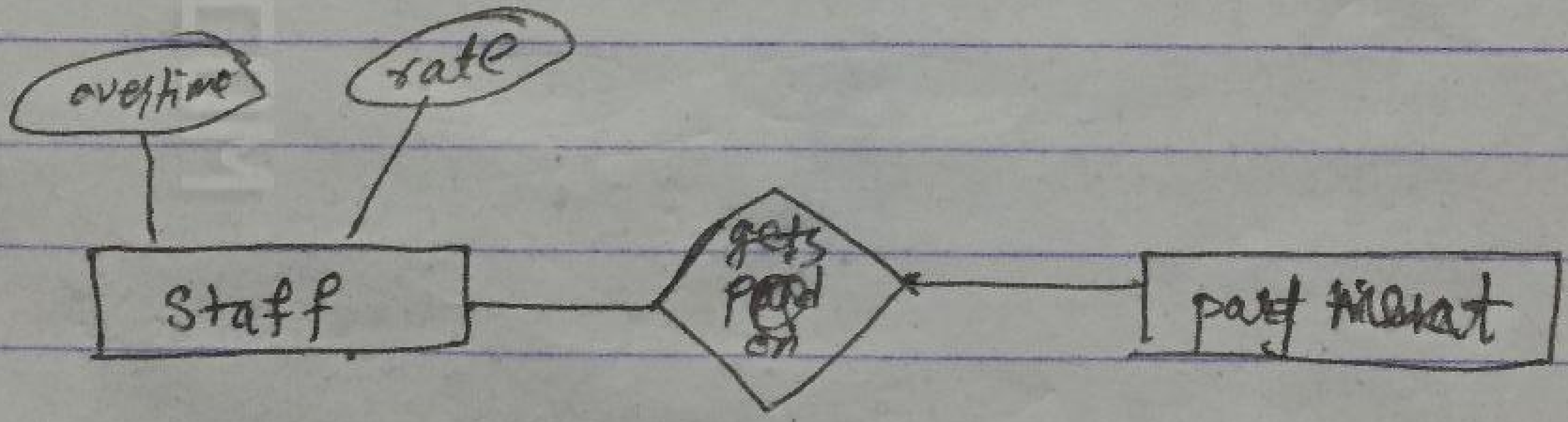
2070 Magh

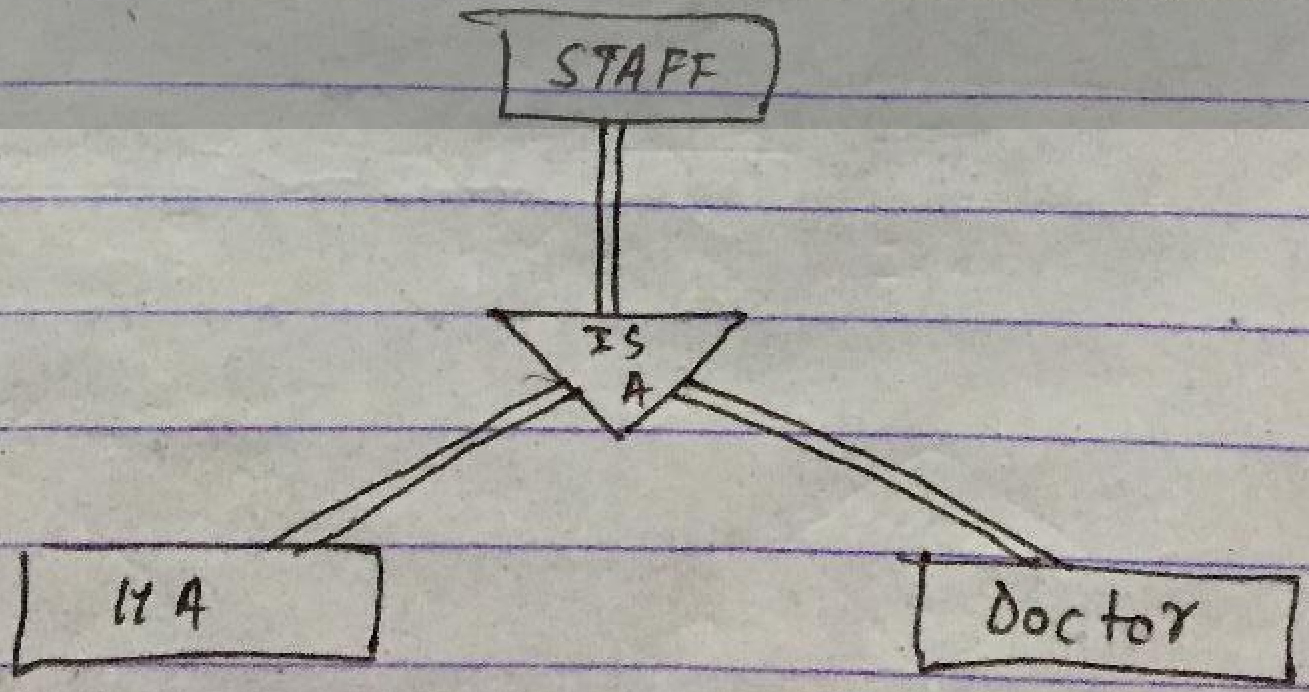


p-no
desc
cost
Product
1*
comprises of
3*
Component
no.
make
c-no
desc
measure

2072 Ashwin


HA
attend
Doctor
assign to
Patient
rec treatment
takes
Drugs
no. of time
length
treated in
associated to
ward


overtime
rate
Staff
~~gets~~ paid on
pay ~~[illegible]~~


STAFF
IS A
HA
Doctor

Chapter-3

Relational Algebra

Relational Algebra

Relational algebra defines a set of operations on relations with the use of algebraic operations. It operates on one or two relations and return a relation as output.

The fundamental operations used are :-

a) Selection
b) Projection
c) Union
d) Set difference
e) Cartesion product
f) Rename
g) Set intersection
h) Natural join
i) Assignment

Select Operation

→ It selects tuples that satisfy a given predicate.
→ It is denoted by 'σ'.
→ Predicates appear as a subscript to 'σ'.
→ Argument relation is in parenthesis after 'σ'.

Syntax : $\sigma_{predicate}$ (Argument Relation)

Eg: To select all the tuples from instructor table whose salary is greater than 10000;

$\sigma_{salary > 10000}$ (instructor)

Project Operation

→ It allows to produce the relation.

→ It is unary operation that returns argument relation [illegible] with certain attributes left out.

→ Duplicate rows are eliminated.

→ It is denoted by 'Π'.

→ These attributes that we wish to appear is listed as a subscript.

→ Argument relation is written in parenthesis.

Syntax :- $\Pi_{attributes}$ (Argument relation)

Union Operation

→ It allows union of two relations as in set theory.

→ It is denoted by '$\cup$'

→ Duplicate values are eliminated.

→ The operation $r \cup s$ is valid if it holds:

1) The relations r and s must have same number of attributes.

2) The domain of i^{th} attribute of r and i^{th} attribute of s must be same, for all i.

Set Difference Operation

→ It is denoted by '−'

→ It allows to find tuples that are in one relation but not in another.

Cartesian Product Operation

→ It is denoted by '×'

→ It allows to combine informations from any two relations.

→ It associates every tuple of one relation with every tuple of another relation.

Rename Operation

→ It gives a name to the result of relational-algebra expressions.

→ It is denoted by 'ρ'.

Syntax : $\rho_x(E)$ → result of E with name x.

$\rho_{x(A_1, A_2, \dots A_n)}(E)$ → result of E with name x and attributes renamed as $A_1, A_2, \dots A_n$.

Set Intersection Operation

→ It is denoted by '$\cap$'

→ It can be replaced by $r - (r - s)$ for $r \cap s$.

Natural Join Operation

→ It combines certain selections and cartesian product into one operation.

→ It is denoted by '$\bowtie$'

Assignment Operation

→ It is denoted by '$\leftarrow$'

→ It assigns parts of relational-algebra expression to temporary relation variables.

Generalized Projection

$$\Pi_{F_1, F_2, \cdots F_n}(E)$$

where, F = arithmetic expression

Aggregation

$$_{G_1, G_2, \cdots G_n}\mathcal{G}_{F_1(A_1), \cdots F_n(A_m)}(E)$$

where, F_i = aggregate function {sum, count, avg, min, max}

G = on which to group

Precedence Order

1) σ, Π, ρ
2) $\times, \bowtie$
3) $\cap$
4) $\cup, -$

8) employee (e-no, name, address)
project (p-no, p-name)
work (e-no, p-no)
part (part-no, part-name, qty, size)
use (e-no, p-no, part-no, number)

1) Print names of employees working on project named "DBMS".

$$\Pi_{name} \left(employee \bowtie \left(\sigma_{p\_name = \text{"DBMS"}} (project) \bowtie work \right) \right)$$

2) Print names of employees not working in any project.

$$\Pi_{name} (employee) - \Pi_{name} (employee \bowtie work)$$

3) List names of employees and projects with which employees are associated with and for which they have used no part so far.

$$\Pi_{name,\, p\_name} \left(work - \Pi_{e\_no,\, p\_no} (use) \bowtie employee \bowtie project \right)$$

4) List names of projects such that every employee working on these projects has used a part "bolts" with size 6.

~~Π_{p_name} (project work, ⋈ (use ⋈ $\sigma_{part\_name = \text{"bolts"} \wedge size = 6}$ (part))~~

$A = \Pi_{e\_no,\, p\_no} \left(\sigma_{part\_name = \text{"bolts"} \wedge size = 6} (part) \bowtie use \right)$

$B = work \bowtie \Pi_{p\_no} (A)$

$C = B - A$

$D = \Pi_{p\_no} (A) - \Pi_{p\_no} (C)$

$N = \Pi_{p\_name} (project \bowtie D)$

5) List part no and part name used in both projects "DBMS" and "MIS".

$\Pi_{part\_no, part\_name}(\sigma_{p\_name = "DBMS"}(project) \bowtie part) \cap$

$\Pi_{part\_no, part\_name}(\sigma_{p\_name = "MIS"}(project) \bowtie part)$

6) List emp name working in both project DBMS and MIS

$A = \Pi_{name}(\sigma_{p\_name = "DBMS"}(project) \bowtie work \bowtie employee)$

$B = \Pi_{name}(\sigma_{p\_name = "MIS"}(project) \bowtie work \bowtie employee)$

$C = A \cap B$

) Acc (<u>a_no</u>, b_name, baln)
Branch (<u>b_name</u>, b_city, assets)
Customer (<u>c_name</u>, c_street, c_city)
Loan (<u>l_no</u>, b_name, amt)
Depositor (c_name, a_no)
Borrower (c_name, l_no)

1) Count number of accounts in each branch.

$$_{b\_name}\mathcal{G}_{count\text{-}distinct(a\_no)}(acc)$$

2) Delete all loans less than $1000 in amount.

$$loan \Leftarrow loan - (\sigma_{amt<1000}(loan))$$

3) Increase loan amount by 10% whose amt is greater than 10000.

$$\Pi_{l\_no, b\_name, amt*1.1}(\sigma_{amt>10000}(loan))$$

Q) emp (eid, name, add, sup_eid)
dep (dep_id, name)
project (pid, title, dep_id)
work (eid, pid, hours)

1) List ~~d~~name of all employees from computer dept. along with name of their supervisor.

$A = \rho_{\bar{A}}(\Pi_{emp.name, sup\_eid}(\sigma_{dep.name = \text{"computer"}}(dep \bowtie emp)))$

$B = \Pi_{emp.name}(emp \bowtie \Pi_{sup\_eid}(A))$

$N =$

2) Delete projects belonging to Electrical dept.

$project \Leftarrow project - \Pi_{pid, title, dep\_id}(project \bowtie \sigma_{name = \text{"Electrical"}}(dep))$

3) Find total no of projects from each department, along with dep name

$({}_{dep\_id}\mathcal{G}_{count\text{-}distinct(pid)}(project)) \bowtie dep$

4) Find ~~total no of p~~ name of all employees who work on "Network" project for more than 15 hours.

$\Pi_{name}(name \bowtie \sigma_{title = \text{"Network"} \wedge hours \geq 15}(project \bowtie work))$

Chapter-3

SQL Query

Q) Consider database schema

Account (acc-no, branch-name, balance)

Branch (branch-name, branch-city, assets)

Customer (customer-name, customer-street, customer-city)

Loan (loan-no, branch-name, amount)

Depositor (customer-name, acc-no)

Borrower (customer-name, loan-no)

Create database and list existing databases

```
CREATE  DATABASE  bank;
SHOW  DATABASES
```

Use database bank for above schema

```
USE  bank;
```

Create tables

```
CREATE  TABLE  account
   ( acc_no  int, not null
     branch_name  varchar (30),
     balance   real,
    primary key (acc_no));

CREATE  TABLE  branch
   (branch_name  varchar (30)  not null,
    branch_city  varchar (30),
    assets  real,
    primary key (branch_name));
```

To list all the tables

```
SHOW TABLES;
```

To insert values into table

```
INSERT INTO account VALUES (123, "Dharan", 12000);
```

Account

acc-no	branch-name	balance
123	Dhr	10000
153	Dhr	12000
157	ktm	12000
165	ktm	18000
170	bkt	158000
179	bkt	198000
182	llt	8975

List all customer details, branch details and acc details according to account number

~~select * from customer, branch, account~~

```
select * from account natural join customer natural join
branch natural join depositor;
```

List no. of accounts in each branch.

```
select   branch_name, count(acc_no)     [count (distinct acc_no)]
from     account
groupby  branch_name;
```

List branch name where avg acc balance is more than 50000

select   branch_name, avg (~~acc~~ balance)
from   account
~~group by branch~~
~~where~~
~~where balance > 50000~~
group by branch_name;
having   avg (balance) > 50000;

Increase all accounts with balance over 10000 by 100% and others by 50%.

```
UPDATE  account
SET   balance = 2*balance
WHERE  balance > 10000;
```

```
UPDATE  account
SET   balance = 1.5* balance
WHERE  balance <= 10000
```

List branch city and total assets where asset > 1000 000.

```
select   branch-city, assets
from     branch
where    assets > 1000000;
```

List the name of customers who have acc in dharan branch.

```
select c.customer_name
from customer as c
natural join  account
natural join  depositor
where   branch_name = "dharan";
```

List customer name whose name begins with 'su'.

```
select customer_name from customer where customer_name
    like 'su%' ;
                [order by customer_name desc]
```

→ For one or more characters : %

→ For one character : _

List name and address of customer. (address = ~~city~~ street, city)

```
select customer-name as name, street + ',' + city as address
from customer
```

Syntax

```
1) select  col-name, col-name            select *
   from  table-name;                     from   table-name;

2) select distinct col-name, col-name
   from  table-name;

3) select  col-name
   from  table-name
   where  col-name  operator  value          // condition

                                          (NOT LIKE)
   =, <>, >, <, >=, <= , BETWEEN, LIKE , IN
                                          ↓          ↘
                                       search a    specify multiple
                                       pattern     possible values

4) To order the result by columns
   select  col-name, col-name
   from table-name
   order by  col-name  asc/desc , col-name  asc/desc;
             ──────────────────────────────→
                      priority order

5) Insert into   table-name
   Values (value1, value2, ...);

6) Update  table_name
   Set   col-name1 = val_1 , col-2 = val 2, ...
   WHERE   condition;
```

7> DELETE from table_name
where condition;

8> To create view

```
CREATE VIEW view_name AS
  }
  } General query
  }
```

Chapter - 4
Database Constraints and ~~Optimization~~ Normalization

Integrity Constraints

→ Integrity constraints are the rules enforced in the data so as to assure accuracy and consistency of data in a relational database.

→ It ensures that the changes made to the database by authorized users do not result in a loss of data consistency.

→ There are four types of integrity constraints as follows:-

a) Domain integrity
b) Entity integrity
c) Referential integrity
d) Foreign key integrity

Domain Integrity

→ It states the set of values for an attributes must be a valid definition.

→ An attribute can be defined with:

a) Data types
b) Length
c) Allowance of null value
d) Default value
e) Value range

→ It ensures that a specific attribute will have a right and proper value.

Entity Integrity

→ It states that primary keys can not be null.

→ As primary key is used to identify individual rows in a table, it can not be null and must have proper value.

Referential Integrity

→ It specifies integrity constraints between two tables.

→ It maintains consistency among rows between two tables.

→ The rules are as follows:-

a) A record from a primary table can not be deleted if matching records exist in a related table.

b) A primary key value in a primary table can not be changed if that record has related records.

c) A foreign key value in a related table can not be inserted if it does not exist in primary key of primary table.

d) A null value can be entered in foreign key specifying that the records are unrelated.

Foreign Key Integrity

→ It contains two constraints.

→ Cascade update related field: Whenever there is change in primary key of a row in primary table, the foreign keys are updated in the matching rows in the related tables.

→ Cascade delete related row: Whenever there is deletion of a row in a primary table, the matching rows are automatically deleted in the related table.

Assertions

→ Assertions are the statement that ensures certain conditions must always be true and must remain true, otherwise the database modification is rejected.
→ It is defined independent of the table.
→ It is activated whenever modification of table mentioned in assertion is needed.

```
CREATE ASSERTION rich
CHECK
((SELECT COUNT(S.sid) From Sailors S)
  + (SELECT COUNT(B.bid) From Boats B) < 100);
```

Triggers

→ It is an event-condition-action rules.
→ It is activated by some events.
→ After activation, it checks the condition.
→ If condition is satisfied, the action is performed.
→ Triggers can be used to automate various actions in the database.

Functional Dependency

→ Functional Dependency (FD) is the relationship that exists when one attribute uniquely determines another attribute.

→ Attribute B is functionally dependent on attribute A, denoted by 'A → B' if a value of A uniquely identifies a value of B at any instant of time.

→ Mathematically, let R be relation schema, $\alpha \subseteq R$ and $\beta \subseteq R$. The FD $\alpha \rightarrow \beta$ holds on R iff for any legal relations r(R), whenever any two tuples t_1 and t_2 agree on the attributes α, they also agree on the attributes β.

i.e. $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$

Eg: Consider a relation r(loan-no, amount) with instances as:

1	10000
2	20000
3	10000
4	25000

Here, (loan_no → amount) holds because for any two tuples, whenever t_1[loan_no] = t_2[loan_no] ⇒ t_1[amount] = t_2[amount]

But, (amount → loan_no) does not hold as for two tuples, the criteria is not satisfied.

i.e. t_1 = (1, 10000) and t_3 = (3, 10000)

t_1[amount] = t_3[amount] ⇒ t_1[loan_no] ≠ t_3[loan_no]

→ In A → B ; A is determinant.

B is object of determinant.

Types of FD

→ A FD is trivial, if the object of determinant is a subset of determinant. Mathematically, a FD $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$.

Eg: (customer_name, loan_number) → customer_name

→ A FD is non-trivial, if the object of determinant is not a subset of determinant.

Eg: (student_id, student_name) → marks

→ Full FD is a situation in which all the attributes of composite determinant is necessary to identify its object uniquely.

→ Partial FD is a situation in which ~~a~~ the subsets of the attributes of composite determinant can identify its object uniquely.

Closure of a set of FD

→ Consider F as the set of FD, then closure of F (F^+) is defined as a set of FD that are implied by F.

→ Axioms:

a) If $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (reflexivity)
b) If $\alpha \rightarrow \beta$, then $\gamma\alpha \rightarrow \gamma\beta$ (augmentation)
c) If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (transitivity)
d) If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$, then ~~$\beta$~~ $\alpha \rightarrow \beta\gamma$ (union)
e) If $\alpha \rightarrow \beta\gamma$ and $\alpha \rightarrow \beta$, then $\alpha \rightarrow \gamma$ (decomposition)
f) If $\alpha \rightarrow \beta$ and $\gamma\alpha \rightarrow \delta$, then $\gamma\beta \rightarrow \delta$ (pseudotransitivity)

Closure of attribute sets

→ Consider A as the set of attributes and F be set of FD, then closure of A under F (A^+) is the set of attributes that are functionally determined by A under F.

→ Algorithm:

```
result := A
while (changes to result) do
    for each $\alpha \rightarrow \beta$ in F
        if $\alpha \subseteq$ result then result := result $\cup \beta$
```

→ If α^+ contains all attributes of R, then α is super key.

→ If $\beta \subseteq \alpha^+$, then $\alpha \rightarrow \beta$ holds.

Q) If $R = (A, B, C, D, E, F, G)$, $F = \{A \rightarrow B, ABCD \rightarrow E, EF \rightarrow G\}$. List all possible F^+.

Ans→

Members of F^+ are :-

- $ACD \rightarrow E$ [$A \rightarrow B$ and $(ACD)\,B \rightarrow E$]
  $ABCDF \rightarrow G$ [$ABCD \rightarrow E$ and $(F)\,E \rightarrow G$]

Q) Find $(AG)^+$ if $R = (A, B, C, G, H, I)$ and $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, G \rightarrow I, B \rightarrow H, CG \rightarrow I\}$.

Ans→

result = AG

result = ABG

result = ABCG

result = ABCGH

result = ABCGHI

Canonical Cover

→ A canonical cover of F is a minimal set of FD equivalent to F having no redundant dependencies.

→ Attribute A is extraneous in α if $A \in \alpha$ and F with $\alpha \rightarrow \beta$ logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.

→ Algorithm

```
Repeat
    Use union rule to replace in F ; $\alpha \rightarrow \beta_1$ , $\alpha \rightarrow \beta_2 \Rightarrow \alpha \rightarrow \beta_1\beta_2$
    Find FD with extraneous attribute and if found, delete attribute.
until F does not change
```

Testing if attribute is extraneous

→ To test if $A \in \alpha$ is extraneous in α.

a) Compute $(\{\alpha\} - A)^+$ under F

b) $(\{\alpha\} - A)^+$ contains all attributes of β, A is extraneous.

→ To test if $A \in \beta$ is extraneous in β

a) Compute α^+ under $F' = (F - \{a \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$

b) If α^+ contains A, A is extraneous.

Q) Compute canonical cover: $R = (A, B, C)$, $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$

Ans ⟹

$F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$

1) $A \rightarrow BC$ and $A \rightarrow B$ have same ~~R.H.S~~ L.H.S attribute, so they can be combined using union rule:

$F_c = \{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

2) To check presence of extraneous attribute in $AB \rightarrow C$;

For $A \in AB$;

$(\{AB\} - A)^+$ under $F = B^+$ under F | output = B

$= \{BC\}$ | output = BC

Since all attributes of R.H.S is in $\{BC\}$, A is extraneous.

$\therefore F_c = \{A \rightarrow BC, B \rightarrow C\}$

3) To check presence of extraneous in $A \rightarrow BC$.

For $C \in BC$; $(A)^+$ under $F' = \{B \rightarrow C, A \rightarrow B\} = \{ABC\}$

As, A^+ contains C, C is extraneous.

$\therefore F_c = \{A \rightarrow B, B \rightarrow C\}$

Decomposition

→ Decomposition is the process of decomposing of a relation into a set of relations.

→ While decomposition, the following properties are desirable:

a) Attribute preservation

b) Lossless join decomposition

c) Dependency preservation

d) Lack of redundancy

Lossless Join Decomposition

→ If R is decomposed in R_1 and R_2, then it is called lossless join decomposition if $R_1 \bowtie R_2 = R$.

→ R_1 and R_2 are lossless decomposition of R, if at least one of the functional dependencies is in F^+.

- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

Dependency Preservation

→ A decomposition is dependency preserving with respect to F if union of projections of F_i on each R_i is equivalent to F.

→ $(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$

~~Q) Consider R = (A, B, C, D, E) F = {A → D, B → E, DE → C}~~

Q) Suppose $R = (A, B, C, D, E)$ is decomposed into $R_1(A, B, C)$ and $R_2(C, D, E)$. Is it lossless? Is it depency preserving?
Consider $F = \{A \rightarrow BC,\ CD \rightarrow E,\ B \rightarrow D,\ E \rightarrow A\}$

Ans ⇒

$R = (A, B, C, D, E)$

$F = \{A \rightarrow BC,\ CD \rightarrow E,\ B \rightarrow D,\ E \rightarrow A\}$

$R_1 = (A, B, C)$ $\quad F_1 = \{A \rightarrow BC\}$ } Not dependency preserving

$R_2 = (C, D, E)$ $\quad F_2 = \{CD \rightarrow E\}$

Now,

~~$(R_1 \cap R_2) = \{C\}$~~

	A	B	C	D	E
R_1	α	α	α		
R_2			α	α	α

Lossy

Again,

$F_1 = \{A \rightarrow BC,\ B \rightarrow D\}$ and $F_2 = \{CD \rightarrow E,\ E \rightarrow A\}$

Here, $F_1 \cup F_2 = \{A \rightarrow BC,\ B \rightarrow D,\ CD \rightarrow E,\ E \rightarrow A\}$

So, $(F_1 \cup F_2)^+ = F^+$

Hence, it is dependency preserving.

F_1 ~~[illegible]~~

Q) $R = (ABCDEG)$, $R_1(AB)$, $R_2(BC)$, $R_3(ABDE)$, $R_4(E, G)$
$F = \{AB \rightarrow C,\ AC \rightarrow B,\ AD \rightarrow E,\ B \rightarrow D,\ BC \rightarrow A,\ E \rightarrow G\}$

Ans. $\Rightarrow$

$R_1(AB)$

$F_1 = \{\ \}$

$R_2(BC)$

$F_2 = \{\ \}$

$R_3(ABDE)$

$F_3 = \{AD \rightarrow E,\ B \rightarrow D\}$

$R_4\ \cancel{=}\ (E, G)$

$F_4 = \{E \rightarrow G\}$

$(F_1 \cup F_2 \cup F_3 \cup F_4) = \{AD \rightarrow E,\ B \rightarrow D,\ E \rightarrow G\}$

$(F_1 \cup F_2 \cup F_3 \cup F_4)^+ \neq F^+$

So, not a dependency preserving.

	A	B	C	D	E	G
R_1	α	α		α	α	α
R_2		α	α	α		
R_3	α	α		α	α	α
R_4					α	α

Lossy (No any rows have all α)

Q) R(A,B C) , R$_1$ (AB) , R$_2$ (AC)
F = { A → B , B → C }

Ans ⇒

F_1 = { A → B }
F_2 = { }

∴ Not a dependancy preserving.

	A	B	C
R_1	α	α	
R_2	α	α	α

Lossless

Q) R = (A, B, C, D, E) , R$_1$ = (A, B, C), R$_2$ = (A, D, E)
F = { A → BC , CD → E , B → D , E → A }

Ans ⇒

F_1 = { A → BC }
F_2 = { ~~CD → E~~ E → A }

∴ Not dependency preserving

	A	B	C	D	E
R_1	α	α	α		[illegible]
R_2	α	α	α	α	α

∴ lossless

8) $R(A,B,C,D)$, $R_1 = (ABC)$, $R_2 (CD)$, $F = \{A \to B, B \to C, C \to D\}$

Ans.

$F_1 = \{A \to B, B \to C\}$

$F_2 = \{C \to D\}$

$\therefore$ Dependency preserving

	A	B	C	D
R_1	α	α	α	α
R_2			α	α

$\therefore$ Lossless

Denormalization is preferred when the operations are read mostly databases. and swift response is crucial.

Multivalued Dependencies

→ R be the relation schema where $\alpha \subseteq R$ and $\beta \subseteq R$, the $\alpha \to\to \beta$ denotes multivalued dependency that holds if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exists t_3 and t_4 such that:

$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$

$t_3[\beta] = t_1[\beta]$

$t_3[R-\beta] = t_2[R-\beta]$

$t_4[\beta] = t_2[\beta]$

$t_4[R-\beta] = t_1[R-\beta]$

→ It express a condition among tuples that exists when:

a> more than one many-many relation

b> certain attribute become independent of one another.

c> Their values must appear in all combinations.

Joined Dependency

→ A join dependency denoted by JD$(R_1, R_2, \cdots R_n)$ on R specifies a constraint on states r of R such that every legal state r of R is equal to join of its projections on $R_1, R_2, \cdots, R_n$.

→ $\pi R_1(r) * \pi R_2(r) * \ldots * \pi R_n(r) = r$

Normalization

→ Database normalization is the process of removing redundant data from tables ensuring data dependencies so as to improve storage efficiency, data integrity and scalability.

→ The stages of normalization are:-

$$\begin{tikzcd}
\text{UNF} \arrow[d] \\
\text{1NF} \arrow[d] \\
\text{2NF} \arrow[d] \\
\text{3NF} \arrow[d] \\
\text{BCND} \arrow[d] \\
\text{4NF} \arrow[d] \\
\text{5NF}
\end{tikzcd}$$

Unnormalized Normal Form (UNF)

→ No normalization rules are applied

First Normal Form (1NF)

→ A relational schema R is in 1NF if the domains of all attributes of R are atomic.

→ Ensure each table has primary key and minimal sets of attributes which uniquely identify a record.

→ Eliminate repeating group (attribute that can have more than one value for a primary key)

→ Each attribute must be atomic.

Second Normal Form (2NF)

→ A relation R is in 2NF iff it is in 1NF and every non key attribute is fully dependent on primary key.

→ Remove partial functional dependencies into a new relation.

Third Normal Form (3NF)

→ A relation R is in 3NF iff it is in 2NF and every non key attribute is non-transitively dependent on primary key.

→ R is in 3NF if for all $\alpha \to \beta$ in F^+, at least one holds:

- $\alpha \to \beta$ is trivial
- α is super key for R
- Each attribute in $\beta - \alpha$ is contained in candidate key for R.

→ Remove transitive dependencies into a new relations.

Boyce-Codd Normal Form (BCNF)

→ It involves decompositions involving relations with more than one candidate key, where candidate keys are composite and overlapping

→ A relation R is in BCNF iff every determinant is a candidate key.

→ It is a special case of 3NF.

→ R is in BCNF if for all $\alpha \to \beta$ in F^+, at least one holds:

- $\alpha \to \beta$ is trivial
- α is super key for R

BCNF and 3NF

→ Relation in BCNF has no information repetition but relation in 3NF has information repetition.

(Low Risk ~~of~~ data integrity)

→ Decomposition in 3NF is lossless and dependency preserving but decomposition in BCNF is not dependency preserving.

Fourth Normal Form (4NF)

→ A relation R is in 4NF if for all multivalued dependencies in D^+ of form $\alpha \to\to \beta$, at least one holds:

- $\alpha \to\to \beta$ is trivial
- α is super key for R.

Fifth Normal Form (5NF)

→ A relation R is in 5NF iff it is in 4NF and every join dependency is implied by the candidate key.

Domain Key Normal Form (DKNF)

→ A relation R is in DKNF iff every constraint of R is a logical consequence of domain constraints and key constraints.
→ It is free from any anomalies.

Example

UNF : sale (cust_name, item, ship-add, newsletter, supplier, supplier-phone, price)

Cust_name	Item	Ship-add	Newsletter	supp	supp-phone	price
A	a	bkt	a1	M	AX	100
B	b	ktm	b1	S	BX	200
C	a, b	bkt	a1, b1	M, S	AX, BX	300
D	c	dhr	c1	S	CX	800

1NF

id	cust_name	Item	Ship-add	Newsletter	supp	supp-phone	price
a_1	A	a	bkt	a1	M	AX	100
b_1	B	b	~~bkt~~ ktm	b1	S	BX	200
c_1	C	a	bkt	a1	M	AX	100
c_1	C	b	bkt	b1	S	BX	200
d_1	D	c	dhr	c1	S	CX	300

2NF

id	cust_name	ship-add	newsletter ~~[illegible]~~
a-1			
b-1			
c-1			
c-1			
d-1			

item	~~[illegible]~~	supp	supp-phone	price

id	item

3NF

item	price	supp

supp	supp-phone

4NF

id	cust_name	ship-add

id	newsletter

Chapter-5
Query Processing and Optimization

Query Processing

→ Query processing involves a range of activities involved to extract data from the database.

Steps Involved In Query Processing

1) Parsing and Translation

→ A given human readable query is translated into its internal form.

→ The parser checks the syntax and also verifies the relation names in the query matches relations in the database.

→ The system construct a parse-tree representation of the query.

→ The parse-tree is used to form a relational algebra expression.

2) Query Optimization

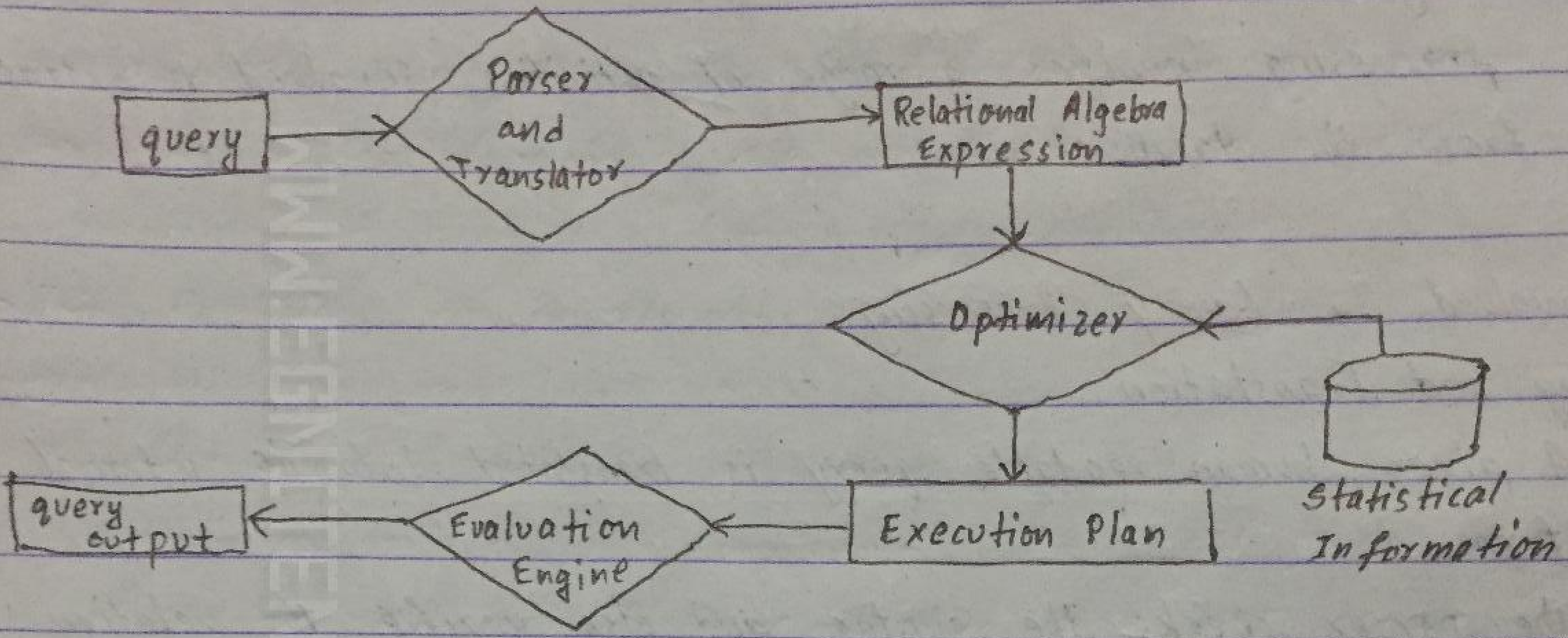
→ The relational algebra expression can be evaluated by many methods or ways.

→ Optimization involves the best way to evaluate the query with lowest cost.

→ Cost is estimated using statistical information from the database catalog.

3) Query Evaluation

→ The evaluation engine takes a selected query evaluation plan, executes the plan and finally provides output to the users.



Example:

Consider a query: select cust-plan
from customer
where cust-plan = "PRO" ;

Relational Algebra

$$\sigma_{cust\_name = "PRO"}(\Pi_{cust\_plan}(customer))$$

Equivalent to

$$\Pi_{cust\_plan}(\sigma_{cust\_plan = "PRO"}(customer))$$

Evaluation Plan

$\sigma_{cust\_name = "PRO"}(\Pi_{cust\_plan}(customer))$

Query Cost Estimation

→ Cost is generally estimated as a response time for a query.

→ Response time depends on disk access, contents of buffer, network communication and so on.

→ Disk access is the predominant cost and is estimated by no of seeks, no. of blocks read or written.

→ The cost for block trasfer is:

$b^* t_T + S^* t_S$ where, b = no. of blocks, S = no. of seeks
t_T = time to transfer one block
t_S = time for one seek.

→ A strategy is choosen based on statistical information for each relation that contains no. of tuples, size of tuples and size of distinct values for a particular attribute.

Query Operations

→ Selection

1) A1 (Linear Search)
   → Search each file block and test all records to determine satisfaction.
2) A2 (Binary Search)
   → Selection is equality an attribute on which file is ordered.
3) A3 (Primary index on candidate key, equality)
4) A4 (Primary index on nonkey, equality)
5) A5 (Secondary index on search key, equality)
6) A6 (Primary index, comparison)
7) A7 (Secondary index, comparison)

→ Sorting

→ quick sort (records completely in main memory)

→ external sort (records are on disk)

→ Join Operation

1) Nested loop join

$r \bowtie_\theta s$

```
for each tuple $t_r$ in r do begin
    for each tuple $t_s$ in s do begin
        test pair $(t_r, t_s)$ to see if condition $\theta$ satisfy
        if they do, add $t_r \cdot t_s$ to the result
    end
end
```

2) Merge Join

→ sort both relations on their join attributes.

→ used for natural joins

3) Hash Join

→ used for natural join

→ A hash function h is used to partition tuples of both relation.

Evaluation of Expressions

→ Materialization

→ Executes a single operation at a time which generates a temporary file that will be used as i/p for next operation.

→ It is easy to implement but time consuming.

→ It walks the parse tree of relational algebra and perform innermost operations first.

→ The result is materialized and becomes i/p for next operations.

→ The cost is the sum of individual operations plus the cost of writing intermediate results to disk.

→ It can always be applied.

→ Pipelining

→ With pipeline, operations form a queue and results are passed from one operation to another as they are calculated.

→ Avoids intermediate temporary relations.

→ Cheaper as no cost for writing results to disk.

→ It is not always possible.

→ In demand driven, system requests next tuple from top level operation, each operation requests next tuple from children operation.

→ In producer driven, operators produce tuples and pass to parents.

Query Optimization

→ It is the process of selecting most efficient query evaluation plan.

→ At relational algebra level, the system attempts to find an expression equivalent to given one but is more efficient.

→ An algorithm can be choosen.

→ Transformation of relational expressions

→ Two relational algebra expressions are said to be equivalent if, on every legal database instance, both expressions generate the same set of tuples.

→ This is the first step of optimization.

→ It generates logically equivalent expressions to the given expression using equivalence rules.

Equivalence Rules

1) $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$

2) $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$

3) $\Pi_{t_1}(\Pi_{t_2}(\Pi_{t_3} \ldots (\Pi_{t_m}(E)))) = \Pi_{t_1}(E)$

4) $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$

$\sigma_{\theta_1}(E_1 \times_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

5) $E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$

6) $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$

7) $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie E_2) = (\sigma_{\theta_1}(E_1)) \bowtie (\sigma_{\theta_2}(E_2))$

8) $\Pi_{L1 \cup L2}(E_1 \ldots_{\theta} E_2) = (\Pi_{L1}(E_1)) \ldots_{\theta} (\Pi_{L2}(E_2))$

9) $E_1 \cup E_2 = E_2 \cup E_1$

$E_1 \cap E_2 = E_2 \cap E_1$

10) $(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$ [Also for $\cap$]

11) $\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - \sigma_\theta (E_2)$ [Also for $\cup$ and $\cap$]

12) $\Pi_L (E_1 \cup E_2) = (\Pi_L (E_1)) \cup (\Pi_L (E_2))$

→ Choice of Evaluation Plan

1) Cost based optimization

→ It explores the space of all query-evaluation plans that are equivalent to given query and chooses one with least estimated cost.

→ Exploring space of all possible plans may be expensive.

→ It guarantees finding of optimal plan.

2) Heuristic optimization

→ System uses heuristics to reduce number of choices to evaluate.

→ A query is transformed by using a set of rules as:

a) Perform selection early

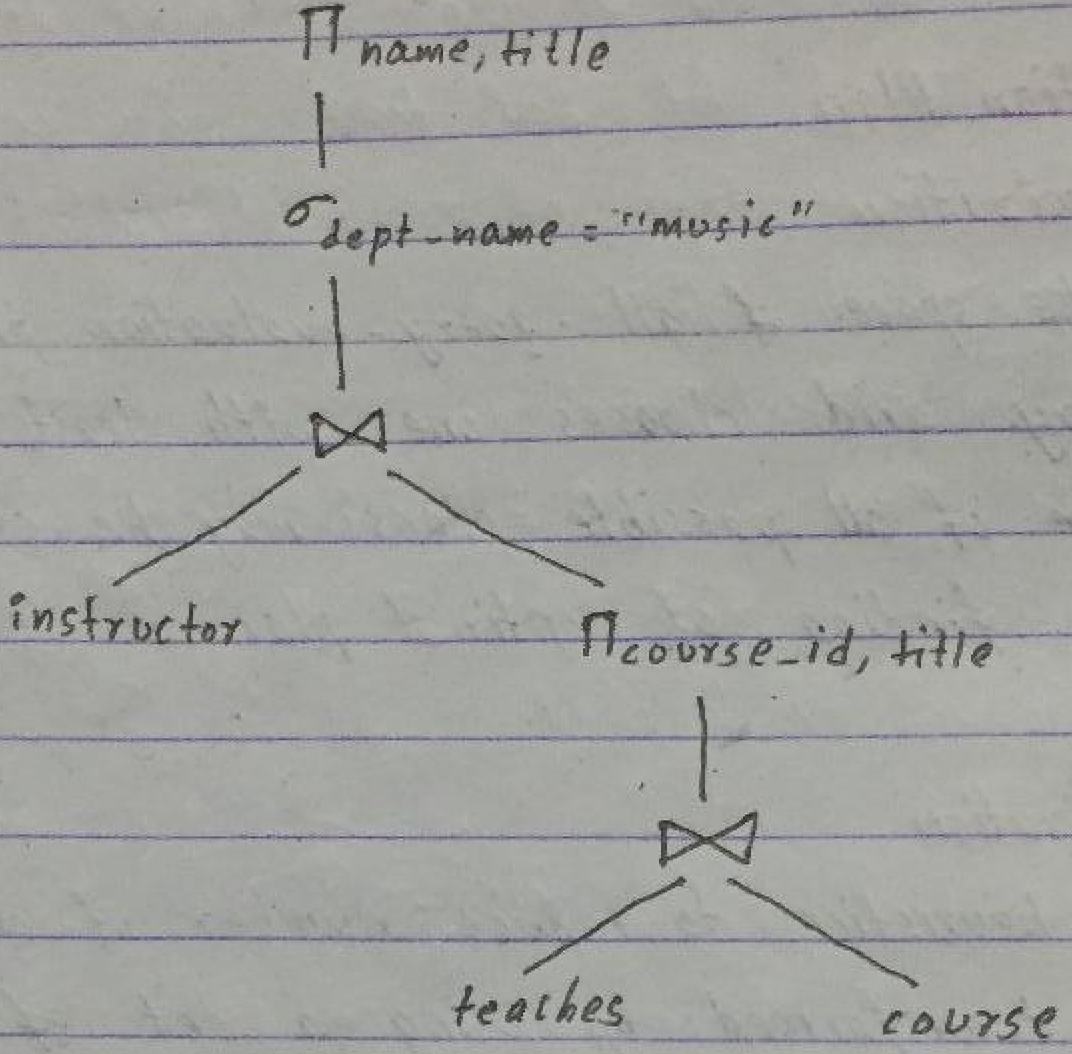
b) Perform projection eary

c) Perform most restrictive selection and join operations before other similar operations.

Q) Optimize the query:

$\Pi_{name, title}(\sigma_{dept\_name = "music"}(instructor \bowtie \Pi_{course\_id, title}(teaches \bowtie course)))$

Ans ⟹

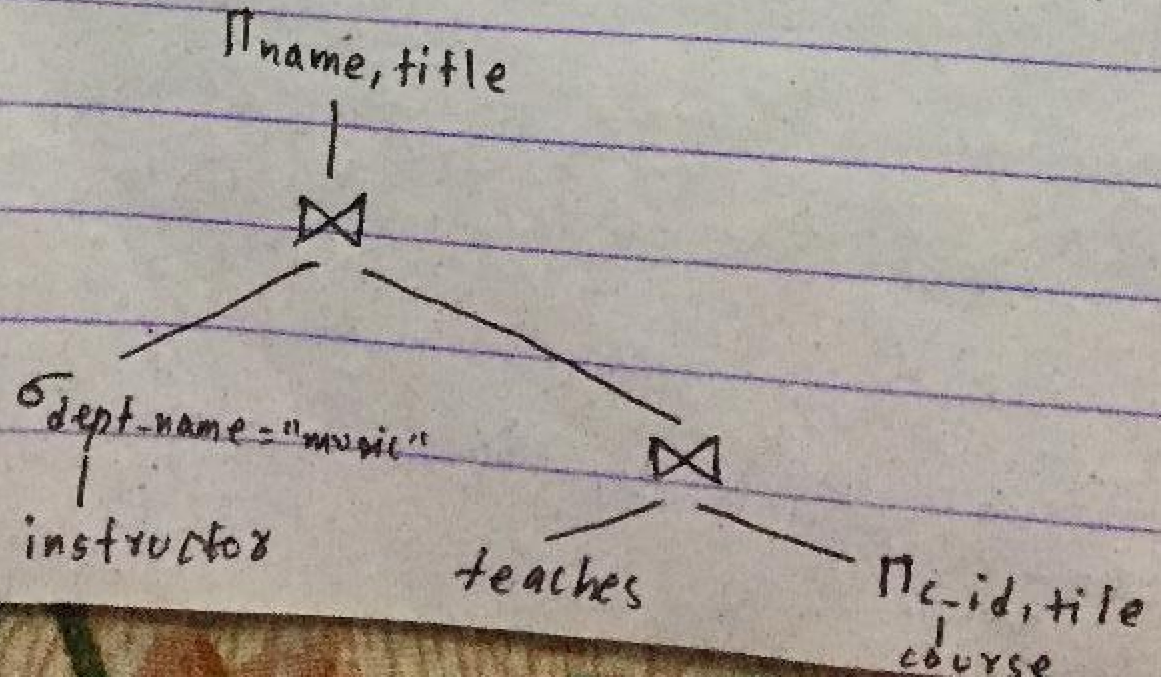


→ Using $\Pi_L(E_1 \bowtie E_2) = E_1 \bowtie \Pi_L(E_2)$

$\Pi_{name, title}(\sigma_{dept\_name = "music"}(instructor \bowtie (teaches \bowtie \Pi_{course\_id, titele}(cours$

→ Using $\sigma_\theta(E_1 \bowtie E_2) = \sigma_\theta(E_1) \bowtie E_2$

$\Pi_{name, titlle}((\sigma_{dept\_name = "music"}(instructor)) \bowtie (teaches \bowtie \Pi_{c\_id, title}(course)))$



Q) Optimize : $\Pi_{name,\ title}(\sigma_{dept\_name=music \wedge year=2009}(instructor \bowtie (teaches \bowtie \Pi_{c\_id,\ title}(course))))$

Ans $\Rightarrow$

→ Using $E_1 \bowtie (E_2 \bowtie E_3) = (E_1 \bowtie E_2) \bowtie E_3$

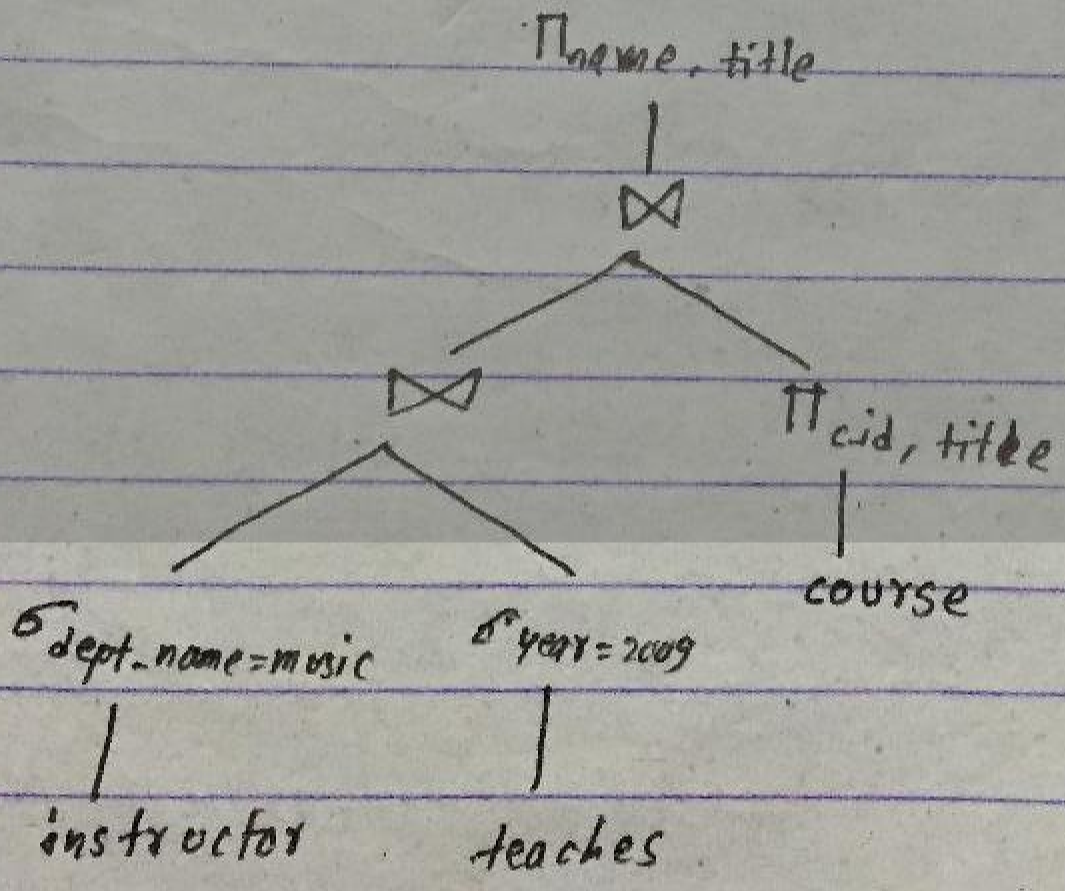
$\Pi_{name,\ title}(\sigma_{dept\_name=music \wedge year=2009}((instructor \bowtie teaches) \bowtie \Pi_{c\_id,\ title}(course)))$

→ Using $\sigma_{\theta_1}(E_1 \bowtie E_2) = \sigma_{\theta}(E_1) \bowtie E_2$

$\Pi_{name,\ title}((\sigma_{dept\_name=music \wedge year=2009}(instructor \bowtie teaches)) \bowtie (\Pi_{c\_id,\ title}(course)))$

→ Using $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie E_2) = \sigma_{\theta_1}(E_1) \bowtie \sigma_{\theta_2}(E_2)$

$\Pi_{name,\ title}(((\sigma_{dept\_name=music}(instructor) \bowtie (\sigma_{year=2009}(teaches))) \bowtie (\Pi_{c\_id,\ title}(course)))$



Query Decomposition

→ It transforms SQL query into relational algebra query.

→ Steps:

a) Normalization

b) Analysis

c) Redundancy elimination

d) Rewriting

Chapter-6

File Structure and Hashing

Physical Storage Media

$$\begin{tikzcd}
\text{Cache} \arrow[d] \\
\text{Main memory} \arrow[d] \\
\text{Flash memory} \arrow[d] \\
\text{Magnetic disk} \arrow[d] \\
\text{Optical disk} \arrow[d] \\
\text{Magnetic tape}
\end{tikzcd}$$

Record Organization

→ Database is stored in a collection of files.
→ Each file is a sequence of records.
→ A record is a sequence of fields.

→ Fixed Length Records

→ The record size is fixed and each file has only one type of records.
→ For different relations, files are different.
→ It is easy to implement.
→ Store record i starting from byte $n*(i-1)$, where n is record size.
→ It is difficult to delete a record from the structure.
→ Sometimes record may cross the boundaries, so two block access is required.
→ For deletion:

a) Move records $i+1, \ldots n$ to $i, \ldots n-1$ but more access to move
b) Link all free records on a free list
c) Store address of deleted record in file header.

→Variable Length Records

→ The variable length record occurs due to:

a> Storage of multiple record types in a single file.

b> Record types allowing variable lengths for fields.

c> Record types allowing repeating fields.

Byte-String Representation

→ A special end of record (⊥) symbol is attached at the end of each record.

→ It is difficult to reuse space occupied by deleted records.

→ The records can not grow.

Slotted Page Structure

→ It is modification of byte-string representation.

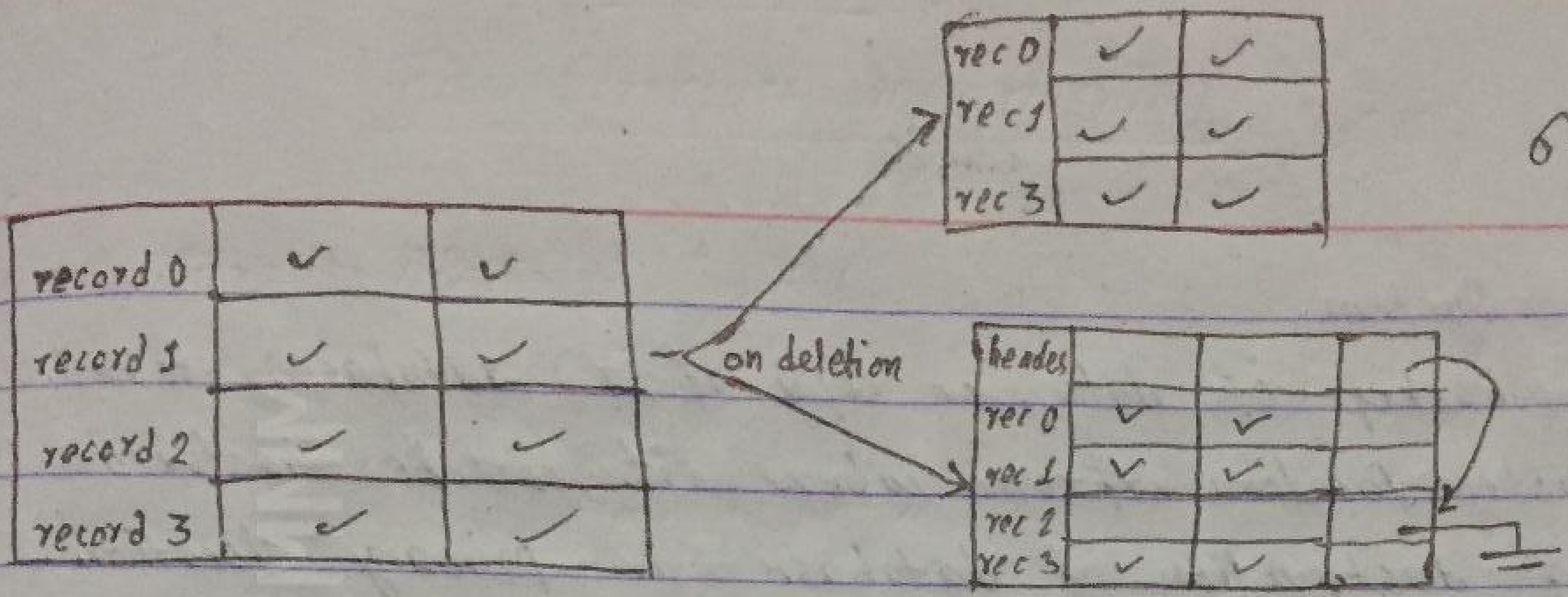
→ Slotted page header contains number of records, end of free space, location and size of each record.

→ Records can be moved to keep them contiguous but header must be updated.

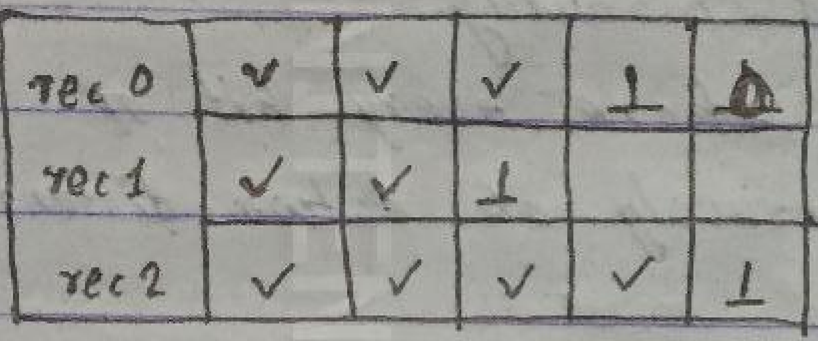
Fixed Length Representation

→ A reserved space is used for fixed length records of known maximum length and unused space are filled with a null.

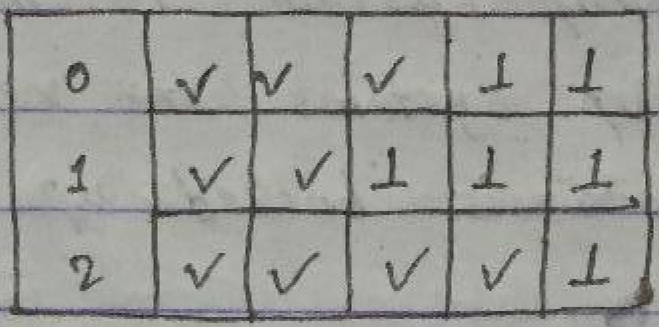
→ A list representation uses fixed length records of a known maximum length



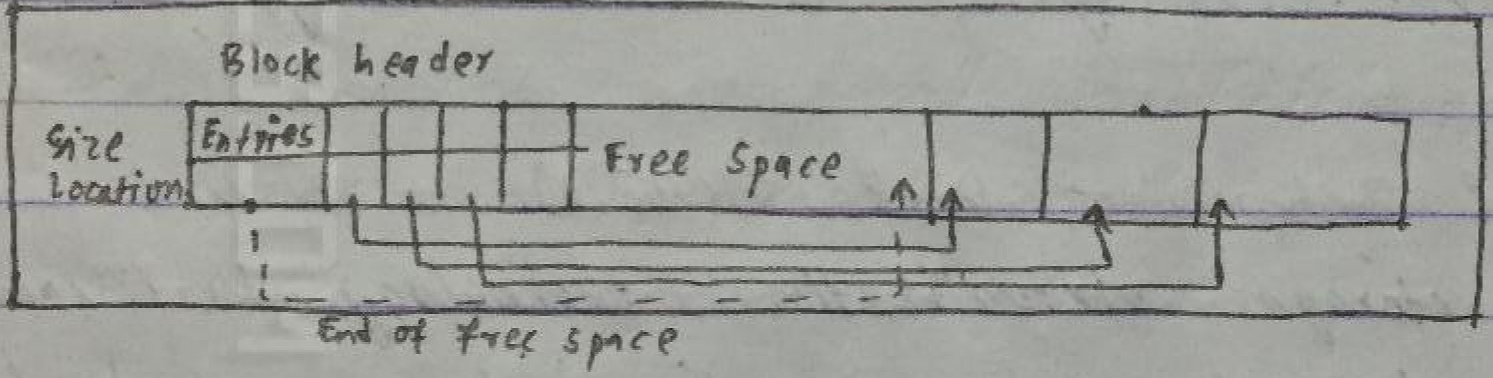
Fixed length record



Byte-string



Reserved space



<u>Organization of Records in Files</u>

1) Heap organization : A record can be placed in any free space.

2) Hash organization : A result of hash function specifies in which block of the file the record should be placed.

3) Sequential organization : The records are stored sequentially. The records are ordered by a search key. For fast retrieval, records are linked together by pointer.

Remote Backup System

→ It is the backup of data or copy of a database stored at a remote location from where it can be restored in case of destruction of database in primary location.

→ It can be offline or online.

→ Offline backup system is maintained manually.

→ Online backup systems are realtime where every bit of real time data is backed up simultaneously at two distant places.

→ As primary database storage fails, the system senses the failure and switches user system to remote storage.

Redundant Array of Independent Disks (RAID)

→ RAID is a data storage virtualization technology that combines multiple physical disk drive components into a single logical unit for the purpose of data redundancy and performance improvement.

→ Data is distributed across drives in several ways called as RAID levels.

→ Each level provides different balance among reliability, availability, performance and capacity.

→ RAID Levels

1) RAID 0 :

→ It consists of stripping without mirroring or parity.
→ It's capacity is the sum of capacities of disks in a set.
→ It can not handle disk failure.
→ Stripping distributes file contents equally among all disks in a set that enables concurrent operations.

2) RAID 1 :

→ It consists of data mirroring.
→ Data is written identically to more drives.
→ Any operation can be serviced by any drive in the set.
→ The one that access data first services the request.
→ Write performance is very slow.

3) RAID 2 :

→ It consists bit level stripping with Hamming code parity.
→ Each sequential bit is on different drive.

4) RAID 3 :

→ It consists of byte level stripping with dedicated parity.

5) RAID 4

→ It consists of block level stripping with dedicated parity.
→ It supports I/O parallelism.
→ I/O operation does not need to spread across all drives.

6) RAID 5

→ It consists of block level stripping with distributed parity.

→ If one drive fails, subsequent read can be calculated from distributed parity.

→ It is susceptible to system failures.

7) RAID 6

→ Block level stripping with double distributed parity.

→ Provides fault tolerance up to two failed drives.

Indexing and Hashing

→ Indexing is a mechanism to speed up access to desired data.

→ A index file consists of records with search key and pointer.

→ Search key is the attribute used to look up records in a file.

→ The record evaluation metrics are:-

a) Access type (Finding records with specified attribute value)

b) Access time

c) Insertion time

d) Deletion time

e) Space overhead (Additional space occupied by index structure)

Ordered Index

→ The index entries are stored sorted on the search key value.

→ Primary Index / Clustering Index

→ The index whose search key specifies sequential order of file.
→ Search key is usually a primary key.

Dense Index

→ Index record appears for every search key value in the file.
→ Index record contains search key and pointer to the first data record with that value.
→ Other records with that value are stored sequentially in a file.

Sparse Index

→ Index records appear for some search key value only.
→ Applicable when records are sequentially ordered on search key.
→ To locate a record with search key value K:
  a) Find index record with largest search key value < k.
  b) Search file sequentially from the record pointed by index pointer
→ It requires less space and less maintenance overhead.
→ It is slower.

Multilevel Index

→ Treat primary index kept on disk as a sequential file and construct a outer sparse index on it.
→ All level of indices must be updated on each operation from the file.

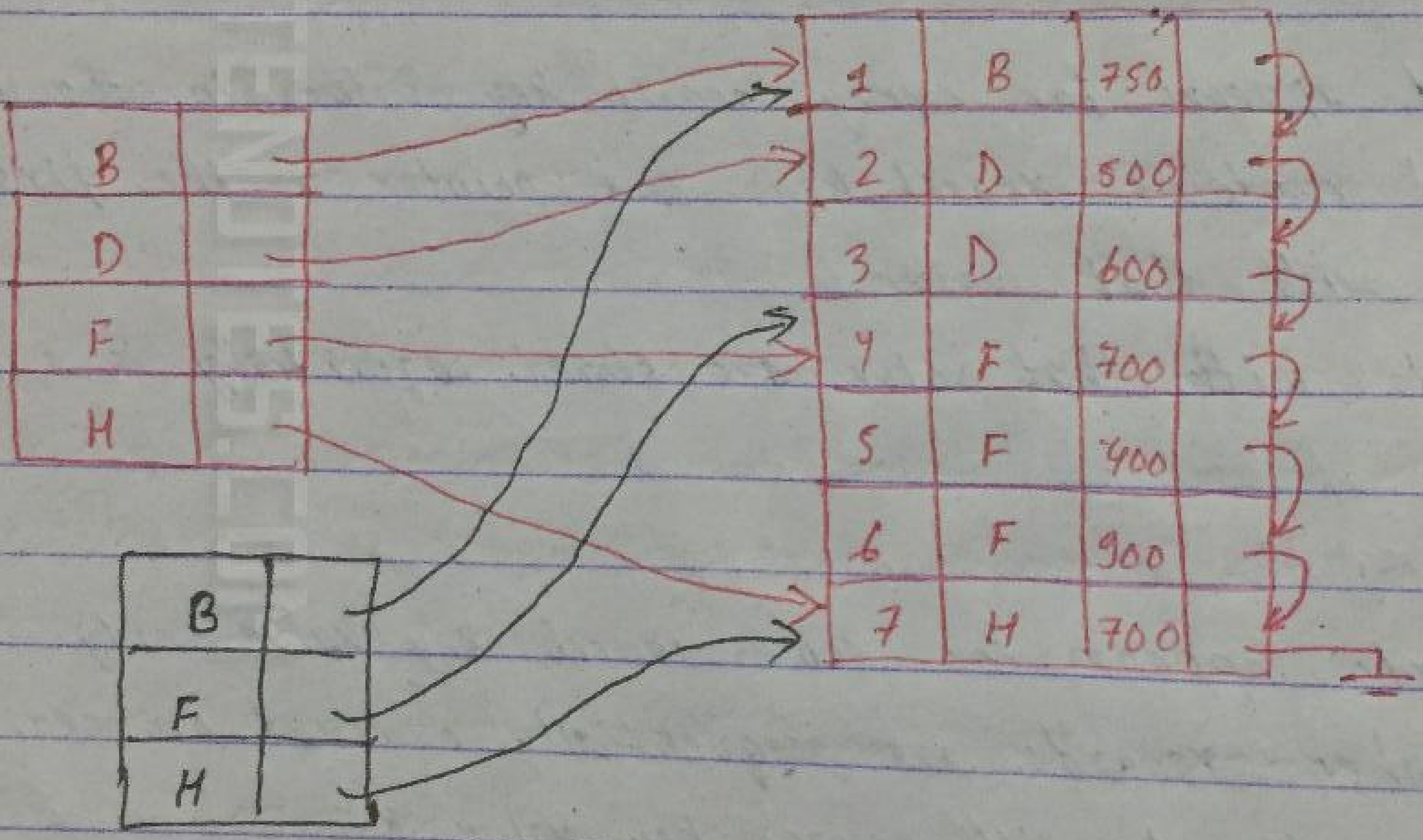
→ Secondary Index

→ The index whose search key specifies order different from the sequential order of the file.

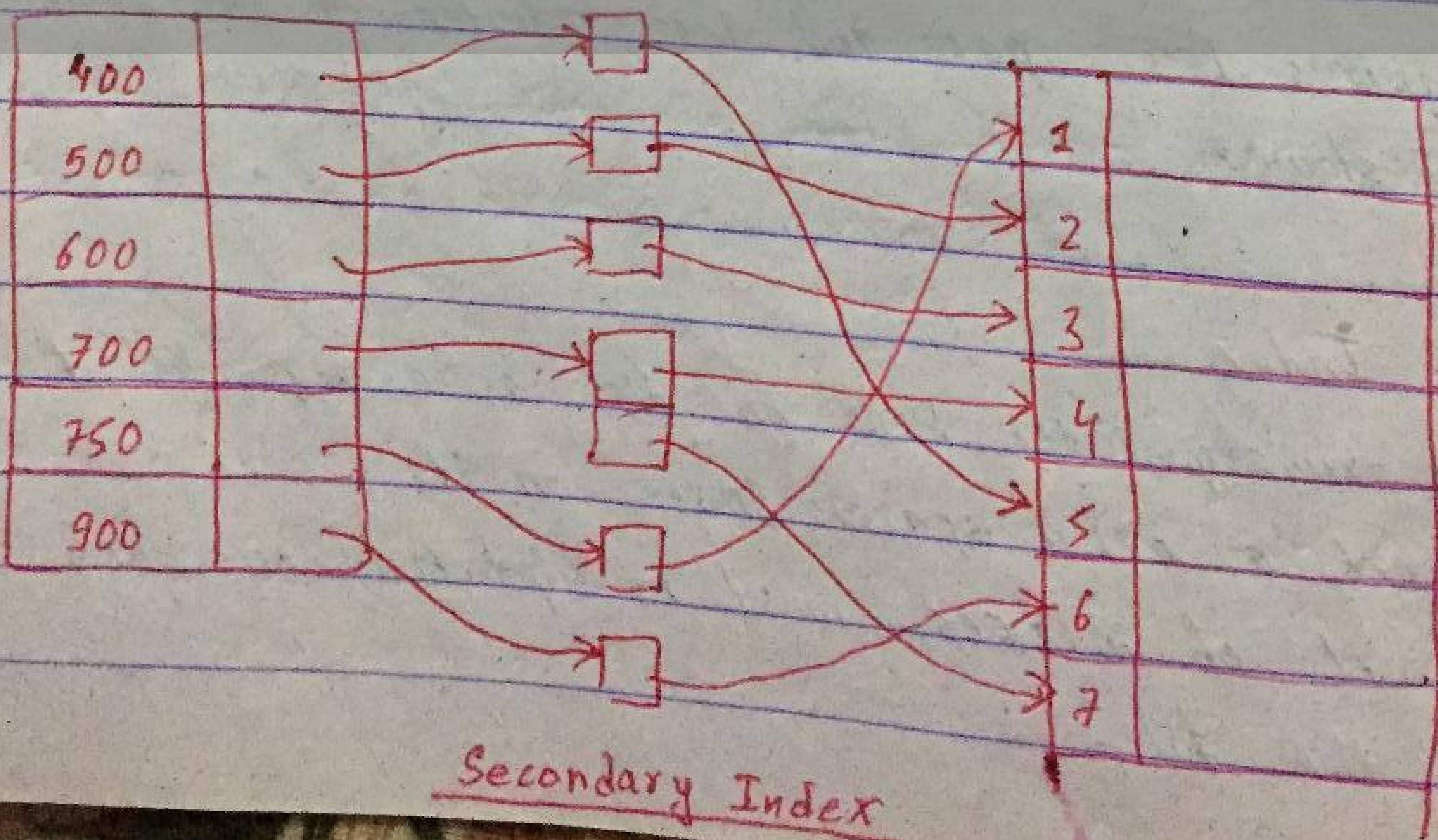
→ It must contains pointers of all the records.

→ Index record points to a bucket that contains pointers to all actual records with that search key value.

→ It must be dense.



Primary Index



Secondary Index

Hashing

→ Bucket is a unit of storage containing one or more records.

→ In hashing, we obtain the bucket directly from search key value using a hash function.

→ Hash function h is a function from the set of all search key values K to the set of all bucket address B.

→ It can be used for access, insertion as well as deletion.

→ Entire bucket has to be searched sequentially to locate a record.

Types of hash functions

→ A hash function is worst if it maps all search key values to the same bucket.

→ A hash function is uniform if it maps same number of search key values to each bucket.

→ A hash function is random if it maps equal number of records to each bucket.

Bucket Overflow

→ Bucket overflow is the condition resulted due to insufficient buckets and skew.

→ To remove bucket overflow, $nB > nr/fr$

→ Skew occurs due to non-uniform distribution ~~of~~ key values by a hash functions and multiple records with same search key value.

→ Overflow buckets are used to handle it.

→ Closed hashing is the chaining of overflow buckets with the buckets in a linked list.

Hash Index

→ A index structure that organizes search keys with their associated record pointers into a hash file structure is called hash index.

→ It is a secondary index.

Static Hashing

→ A hash function h maps search key values to a fixed set of bucket address B.

→ The main problem arises when databases grow or shrink with time.

→ If initial no. of buckets is too small, due to more overflows as file grows, performance is reduced.

→ If file shrinks or if space is allocated for growth, it will waste space.

→ Periodic reorganization with new hash function can solve it but is expensive.

Dynamic Hashing / Extendable Hashing

→ The set of keys can be varied and address space is mapped dynamically

→ Growth and shrink of database ~~dyna~~ do not affect the organization

→ Hash function can be modified dynamically.

→ Extendable hashing is a form of dynamic hashing.

→ Keys stored in bucket and each bucket can hold fixed item.

→ Index is an extendible table.

→ $h(x)$ hashes a key value x to a bit map.

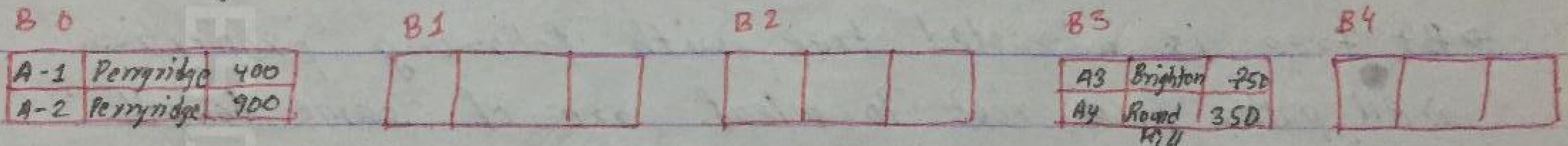
→ Extra indirection level to find desired record
→ Bucket address table itself may become big.
→ changing size of bucket address table is expensive.

→ Minimal space overhead
→ Once a bucket is full, it is splitted.
→ No reorganization is required when data are added to or deleted from the file.

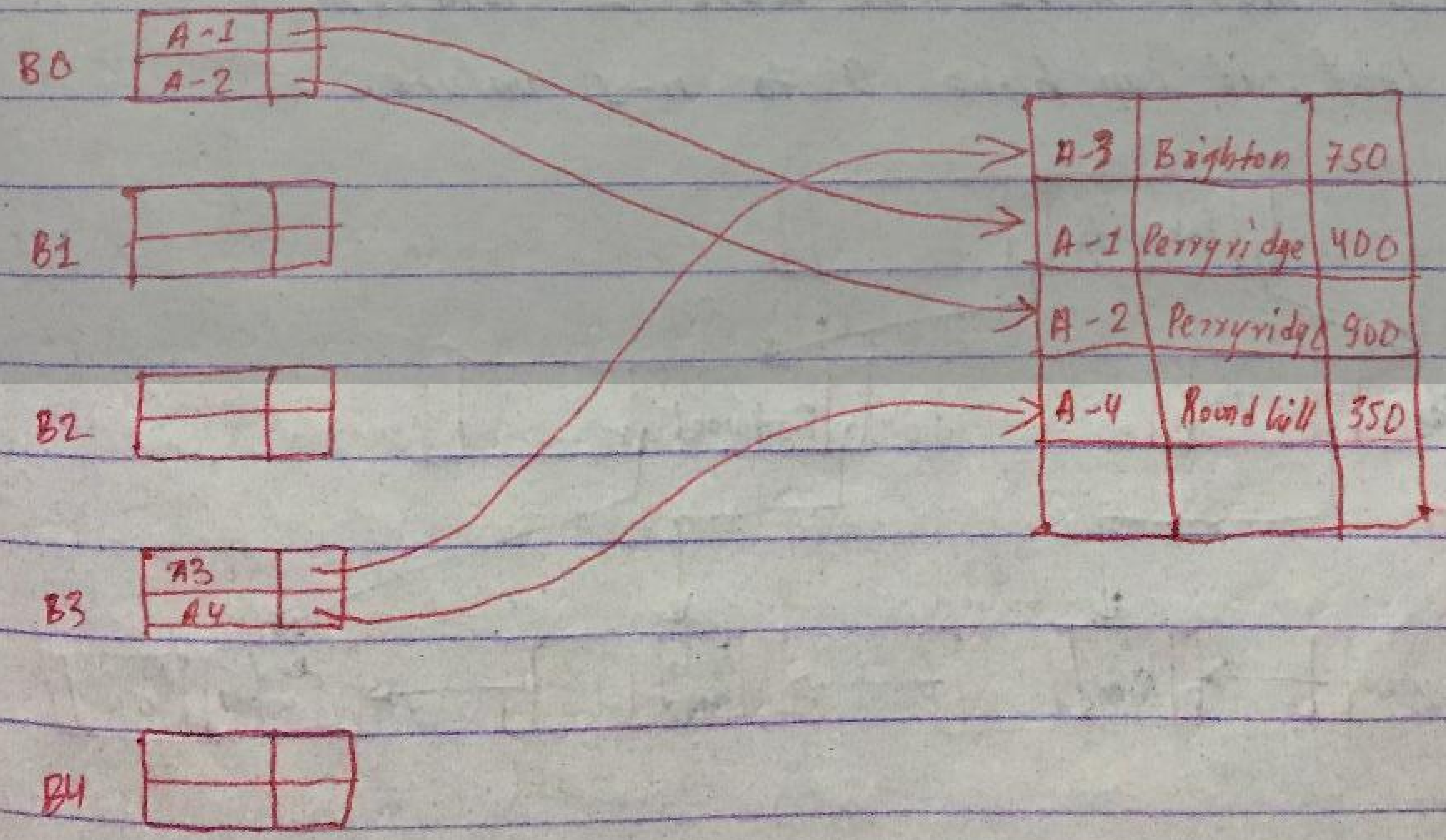
Example of hashing

Consider there are 5 buckets, binary of ith character is integer i and hash function returns sum of binary of characters modulo 5.

Then, h (Perryridge) = 0
h (Round Hill) = 3
h (Brighton) = 3



Example of hash index



→ Hashing is better to retrieve records with specified value of key
→ Ordered index is better for range queries.

B+ Tree Index

→ It is alternative to index-sequential files.

→ Advantages

a) Automatic reorganization of insertion and deletion.

b) Performance is independent of entire file reorganization.

→ Disadvantages

a) As file grows, overflow blocks are created.

b) Periodic reorganization of entire file is required.

c) Space overhead.

→ B+ tree is a rooted tree with following properties.

a) All paths from root to leaf are of same length.

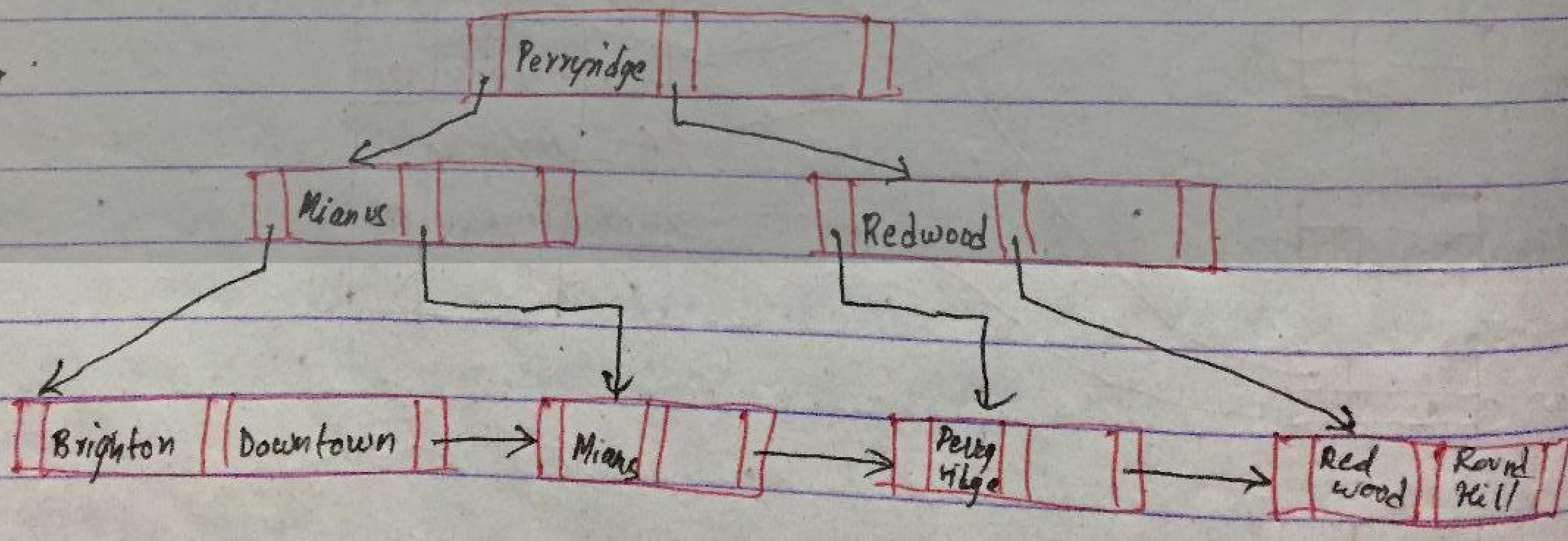
b) Each non-root or non-leaf node has n/2 ~~and~~ to n children.

c) Leaf node has between (n-1)/2 and n-1 values.

d) Non-leaf root must have at least 2 children.

e) If root is leaf, it can have 0 to n-1 values.

→



→ A typical node contains K_i (search key value) and P_i (pointer to children for non-leaf nodes or pointer to records of records for leaf nodes)

→ $K_1 < K_2 < \dots < K_{n-1}$

Chapter - 7
Transaction Processing and Concurrency Control

Transactions

→ Transaction is a single logical unit of operation which on collection performs some meaningful actions.

→ The system should ensure proper execution of transactions.

→ Either the entire transaction executes or none of them executes.

→ The system also manages concurrent execution of transactions.

→ Each transaction access and possibly updates various data items.

ACID Properties

1) Atomicity :

→ Either the entire transaction completes or none of it completes.

→ Transaction is indivisible.

2) Consistency:

→ Execution of transaction must preserve database consistency.

→ If a transaction runs from a consistent database state, the database must be in consistent state after the end of transaction.

3) Isolation :

→ A transaction should not appear its effects on other concurrent transactions.

→ Each transaction must operate properly without interference from concurrent operations.

→ For a pair T_i and T_j, it appears to T_i that either T_j finished execution before T_i started or T_j started execution after T_i completes.

4) Durability:

→ The changes made to the database by a transaction must be persistent even if the system fails or crashes.

Transaction Model

$$\begin{tikzcd}
 & \text{Partially committed} \arrow[r] \arrow[dd] & \text{Committed} \\
\text{active} \arrow[ru] \arrow[rd] & & \\
 & \text{Failed} \arrow[r] & \text{Aborted}
\end{tikzcd}$$

1) Active state: The transaction is executing in this state.
2) Partially committed: The final statement has been executed.
3) Committed: The transaction is completed successfully.
4) Failed: Normal execution can not proceed.
5) Aborted: Transaction is rolled back and database is restored.

Concurrent Execution

→ The system allows multiple transactions to run concurrently.
→ It increases CPU and disk utilization increasing throughput.
→ It reduces the average response time of transaction.
→ Concurrency control schema must be formulated to control interaction among concurrent transactions to prevent database consistency.

→ Schedule

→ Schedule is the sequence of instructions that specify chronological order in which instructions of concurrent transactions are executed.

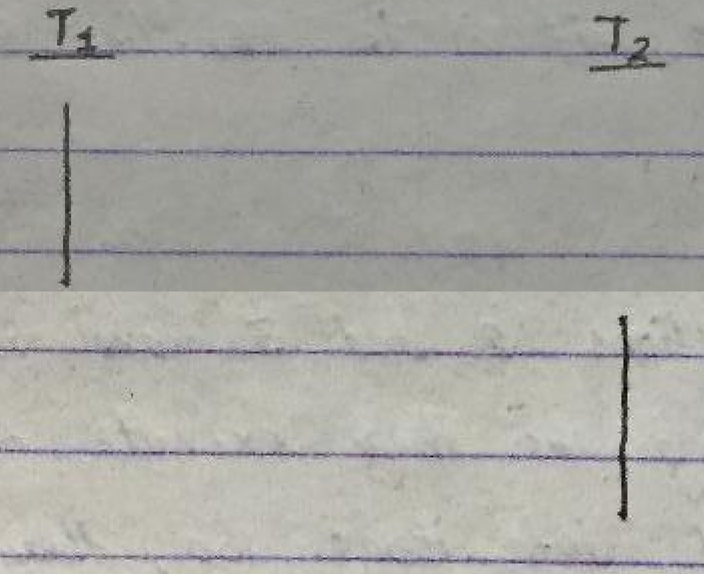
→ It must preserve the consistency of the database.

Consider two transactions T_1: transfer 50 from A to B and T_2: transfer 10% balance from A to B defined as:

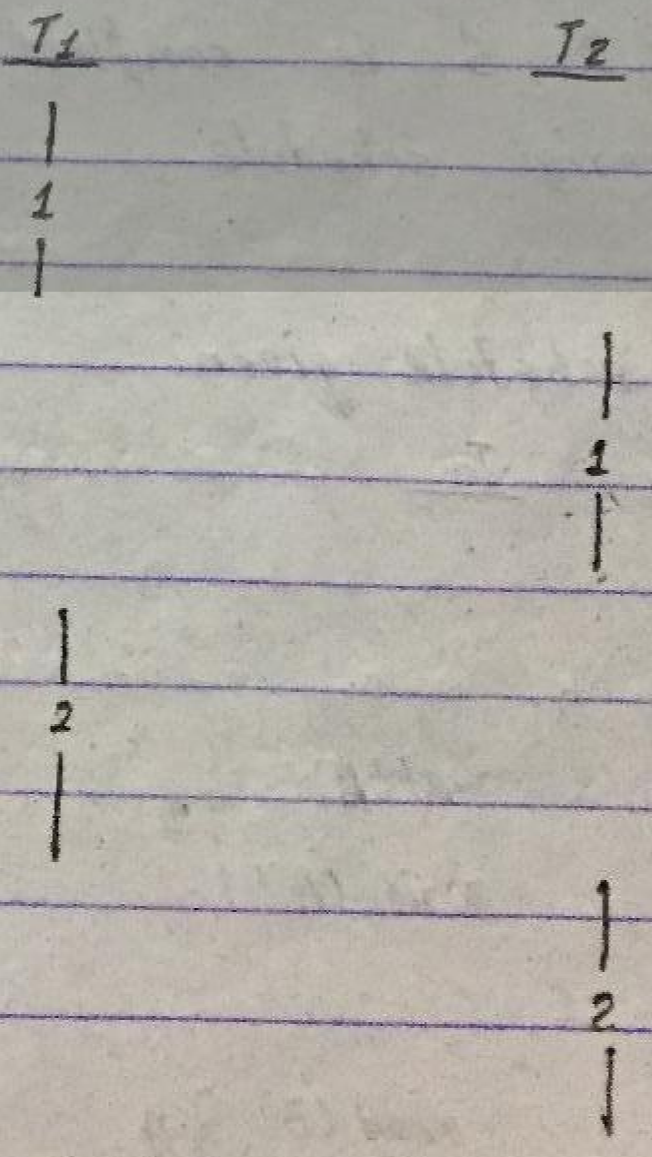
```
T1: read (A);      }
    A := A-50;     } 1
    write (A)      }
    read (B);      }
    B := B +50;    } 2
    write (B)      }

T2: read (A);               }
    temp := A * 0.1;        }
    A := A - temp;          } 1
    write (A);              }
    read (B)                }
    B := B + temp;          } 2
    write (B)               }
```

~~A := A~~

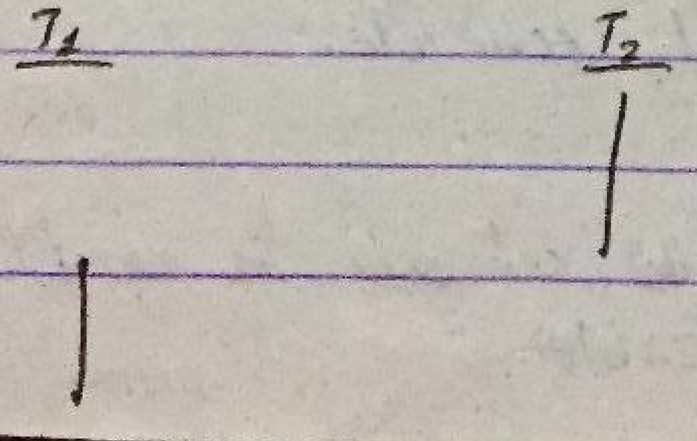
Serial Schedule T_1 followed by T_2



Not a serial schedule



Serial Schedule T_2 followed by T_1



Serializability

→ A schedule is serial if for every transaction participating in the schedule, all operations of a transaction executed consecutively in the schedule.

→ A schedule is serializable if it is not serial but is equivalent to some serial schedule of same transactions.

→ Two schedules are result equivalent if they produce the same final state of the database.

Forms of schedule equivalence

1) Conflict serializability

→ Two actions A_i and A_j executed on same data object by T_i and T_j conflicts, if either one of them is write operation.

→ If A_i and A_j are consecutive non-conflicting actions belonging to T_i and T_j, we can swap A_i and A_j without changing result.

→ If schedule S can be transformed to S' by a series of swap, S and S' are conflict equivalent.

→ A schedule S is conflict serializable, if it is conflict equivalent to a serial schedule.

Consider schedule given

	T_1	T_2	
1	read (A)		
	write (A)		
		read (A)	2
		write (A)	
3	read (B)		
	write (B)		
		read (B)	4
		write (B)	

Here, actions 2 and 3 are non-conflicting as they execute on different data items. So, they can be swapped which results in a serial schedule.

Hence, the schedule is conflict serializable.

Consider schedule given:

	T_3	T_4
1 {	read(Q)	
		write(Q) } 2
3 {	write(Q)	

Here, 1 and 2 are conflicting so can not be swapped. Also, 2 and 3 are conflicting.

Hence, it is $\overset{\text{not}}{\wedge}$ conflict serializable.

→ Test for conflict serializability

1) For each transaction T_i in schedule S, create a node labeled T_i in the precedence graph.

2) For each case in S where T_j executes read(X) after T_i executes write(X), create edge $(T_i \rightarrow T_j)$

3) For each case in S where T_j executes write(X) after T_i executes read(X), create edge $(T_i \rightarrow T_j)$

4) For each case in S where T_j executes write(X) after T_i executes write(X), create edge $(T_i \rightarrow T_j)$

5) S is serializable iff graph has no cycle.

Eg: Conside the transactions:

$T_1 : r_1(x);\ w_1(x);\ r_1(y);\ w_1(y)$

$T_2 : r_2(z) : r_2(y);\ w_2(y);\ r_2(x);\ w_2(x)$

$T_3 : r_3(y);\ r_3(z);\ w_3(y);\ w_3(z)$

Consider schedule: $S_1 : r_2(z);\ r_2(y);\ w_2(y);\ r_3(y);\ r_3(z);\ r_1(x);\ w_1(x);$
$w_3(y);\ w_3(z);\ r_2(x);\ r_1(y);\ w_1(y);\ w_2(x)$

$$\begin{tikzcd}
T_1 \arrow[rr, bend left, "x"] & & T_2 \arrow[ll, "y"'] \arrow[dl, "{y,z}"] \\
& T_3 \arrow[ul, "y"] &
\end{tikzcd}$$

Not conflict serializable

2) View Serializability

→ The two schedules S_1 and S_2 are view equivalent if:

a) For each data item Q, if transaction T_i reads initial value of Q in S_1 then transaction T_i must read initial value of Q in S_2.

b) For each data item Q, if T_i reads Q after T_j writes Q in S_1, then T_i must read Q after T_j writes Q in S_2.

c) For each data item Q, if T_i writes final value of Q in S_1, then T_i must writes final value of Q in S_2.

→ A schedule is view serializable if it is view equivalent to a serial schedule.

<u>Eg:</u> Consider transactions:

$T_1 : r_1(Q)\ ,\ w_1(Q)$

$T_2 : w2(Q)$

$T_3 : w3(Q)$

Schedule: $S : r1(Q), w2(Q), w1(Q), w3(Q)$

Let S' be serial schedule : $S' : r_1(Q), w_1(Q), w2(Q), w3(Q)$

a) T_1 reads initial value of Q in S and S'

b) No condition

c) T_3 writes final value of Q in S and S'.

∴ S is view serializable.

Recoverable Schedule

→ A schedule is recoverable if a transaction T_j reads a data item previously written by transaction T_i, then T_i should commit before T_j commits.

→ Cascading rollback is a ~~situat~~ mechanism that performs a series of transaction rollbacks if a single transaction fails.

→ Cascadeless schedule is the schedule in which for each pair of transactions T_i and T_j such that T_j reads data item previously written by T_i, T_i must commit before T_j reads.

Lock Based Protocol

→ It is a protocol that ensures that when one transaction is accessing a data item, no other transaction can modify that data item with the use of locks.

→ A lock based protocol is a set of rules followed by all the transactions while requesting and releasing locks.

→ The transaction holding the lock on that item can only access that data item.

→ Locks can be in two modes. The transaction holding shared mode lock (lock-S) can only read data item. The transaction holding exclusive mode lock (lock-X) can both read and write data item.

→Lock Compatibility Matrix

	S	X
S	T	F
X	F	F

→ Transaction requests concurrency control manager for locks.
→ A transaction may be granted a lock on an item if requested lock is compatible with locks already held on the item by other transactions.
→ Any no. of transactions can hold shared locks on an item.
→ If lock is not granted, the requesting transaction must wait.

→ Disadvantages
1> Deadlock
2> Starvation

→ Two Phase Locking Protocol
→ Two phase locking protocol ensures conflict serializable schedules.
→ Phase 1 : Growing Phase
a> Transaction may obtain locks.
b> Transaction may not release locks.
→ Phase 2 : Shrinking Phase
a> Transaction may release locks.
b> Transaction may not obtain locks.
→ It results in deadlock.
→ It may cause cascading rollback (Use strict two phase locking)
→ Rigorous two phase locking (holds all locks until commit)

Example

```
T1: lock-S(A);
    read(A);
    unlock(A);
    lock-S(B);
    read(B);
    unlock(B);
    display(A+B)
```

Lock based protocol

```
T1: lock-S(A);   }
    read(A);     }  Growing
    lock-S(B);   }
    read(B);     }
    unlock(B);   }
    unlock(A);   }  Shrinking
    display(A+B);}
```

→ Lock Conversion

- → It is a mechanism for upgrading a shared lock to exclusive lock and downgrading an exclusive lock to a shared lock.
- → Upgrade takes place in growing phase.
- → Downgrade takes place in shrinking phase.
- → It assures serializability.

→ Implementation of Locking

- → Lock manager replies to a lock request by sending lock grant message.
- → In case of deadlock, it sends msg asking for rollback.
- → The requesting transaction waits until its requests is answered.
- → It maintains data structure called lock table.
- → Lock table is in-memory hash table indexed on name of data being locke
- → Lock table also records type of lock granted or requested.
- → New requests added to end of queue.
- → If transaction aborts, all its requests are deleted.

Deadlock

→ Deadlock is the condition in which all transactions in a set are holding locks to some data items and waiting for locks to other data items currently being held by other transactions of same set.

→ Deadlock Prevention

1) Ensure no cyclic waits
   - → All locks should be acquired together. before execution.
   - → Ordering requests for locks.
2) No wait
   - → Wait-die scheme / Rollback [T_i waits only if it has timestamp smaller than that of T_j, otherwise rolled back]
   - → Wound-wait scheme / Preemption
   - → Timeout based scheme.

→ Deadlock Detection

→ Deadlock can be described as a wait-for graph $G = (V, E)$.
→ V is all the transactions.
→ E is eldges showing $T_i \rightarrow T_j$
→ If $T_i \rightarrow T_j$ is in E, there is directed edge from T_i to T_j, implying that T_i is waiting for T_j to release data item.
→ The system is in deadlock iff wait-for graph has a cycle.

Chapter-8
Crash Recovery

Crash Recovery

→ Crash recovery is the scheme in a database system that can restore the database system to the consistent state that existed before the crash.

→ The time for recovery must be minimal.

Failure Classification

1) Transaction Failure

→ The transaction can not continue because of some internal conditions like data mismatch, overflow, etc. This is logical error.

→ The transaction can not continue as the system enter into an undesirable state like deadlock. This is system error.

2) System Crash

→ It is the failure caused due to hardware malfunctioning or software bugs that do not affect data of non-volatile storage.

→ It is called fail-stop assumption.

3) Disk Failure

→ It is the failure caused due to head crash or data transfer failure.

→ Multiple backups are used to recover from this failure

Log Based Recovery

→ Log is a sequence of log records, recording all the database update activities.

→ When transaction starts, it registers itself by writing a <T_i start> log record.

→ Before T_i executes write (X), a log record <T_i, X, V_1, V_2> is written, where V_1 is value of X before write and V_2 after write.

→ When T_i finishes, the log record <T_i commit> is written.

→ Deferred Database Modification

→ It records all modifications to the log, but defers all the writes to after partial commit.

→ If system crashes before completion of transaction or if transaction is aborted, the information on the log is ignored.

→ Transaction starts by writing <T_i start> record to log.

→ A write(X) results in log record <T_i, X, V>, where V is the new value of X.

→ The write is not performed at this time.

→ When T_i partially commits, <T_i commit> is written to log.

→ Log records are read and execute deferred writes.

→ The recovery scheme uses following procedures:

1> Redo (T_i) sets the value of all items updated by transaction T_i to the new values which can be found in log.

→ Executing redo() several times must be equivalent to single execution.

→ Redo is necessary iff both <T_i start> and <T_i commit> are in the log.

→ Immediate Database Modification

→ It allows modifications as soon as write operation is executed.

→ It is called uncommitted modifications.

→ Update logs consist of both old value and new value.

→ The recovery scheme uses :-

1) Undo (T_i) restores value of all data items updated by T_i to their old values, going backwards from last log record.

2) Redo (T_i) sets value of all data items updated by T_i to their new values, going forward from first log record.

→ Undo is required if <T_i start> is present but <T_i commit> is absent.

→ Redo is required if <T_i start> and <T_i commit> both are present.

→ Undo is performed first.

Example

<T_0 start>	<T_0 start>
<T_0, A, 950>	<T_0, A, 1000, 950>
<T_0, B, 2050>	<T_0, B, 2000, 2050>
<T_0, commit>	<T_0, commit>
<T_1 start>	<T_1, start>
<T_1, C, 600>	<T_1, C, 700, 600>
→ Redo (T_0)	→ Undo (T_1)
→ No action for T_1	→ Redo (T_0)

→ Checkpoints

→ Searching a log is time consuming.

→ It is unnecessary to redo transactions which already o/p updates to database

→ It does not permit any updates to be performed while checkpoint is in progress.

→ It outputs all modified buffer blocks to disk when checkpoint is performed.

→ It allows streamlining of recovery procedore.

→ If a record commit before <checkpoint> record. During recovery, no redo is necessary on that operation.

→ The system examines the log to find last <checkpoint L> record by searching log backward from the end.

→ For all T_x in T without $\langle T_x$ commit$\rangle$ in the log, execute undo(T_x).

→ For all T_x in T with $\langle T_x$ commit$\rangle$ in the log, execute redo(T_x)

Eg:-

$\langle T_1$ start$\rangle$	
$\langle T_1$, B, 2000, 2500$\rangle$	→ T_1 ignored
$\langle T_1$ commit$\rangle$	→ T_2 redo
$\langle T_2$ start$\rangle$	→ T_3 redo
$\langle T_2$, A, 200, 100$\rangle$	→ T_4 undo
$\langle$Checkpoint $\{T_2\}\rangle$	
$\langle T_2$ commit$\rangle$	
$\langle T_3$ start$\rangle$	
. .	
.	
~~$\langle T_3$ commit$\rangle$~~	
$\langle T_4$ start$\rangle$	

Shadow Paging

→ It is useful if transactions execute serially.

→ The current page table and shadow page table are maintained during the lifetime of a transaction.

→ Shadow page table is stored in non-volatile storage with state prior to transaction.

→ It is not modified during execution.

→ Only current page table is used for access during execution.

→ It may cause commit overhead, data fragmentation and garbage collection.

Advanced Recovery Algorithm

→ It supports high concurrency locking techniques which release locks early.

→ Physical undo can not be performed.

→ Logical Undo

→ It allows undo logically i.e. insertion can be undone by executing deletion.

→ Operation Logging

→ When operation starts, $\langle T_i, O_j$, operation-begin$\rangle$ is logged.

→ When operation completes, $\langle T_i, O_j$, operation-end, $U\rangle$ is logged, where U contains information for logical undo.

→ If operation not completed, physical undo.

→ If operation completed, logical undo.

→ Txn Rollback

Scan the log backwards

→ If log $\langle T_i, X, V_1, V_2 \rangle$ found, perform undo and log a special redo-only log record $\langle T_i, X, V_1 \rangle$

→ If $\langle T_i, O_j, \text{operation-end}, U \rangle$ is found, rollback logically using undo information U.

At the end, generate record $\langle T_i, O_j, \text{operation-abort} \rangle$

→ Skip all log records for T_i until $\langle T_i, O_j, \text{operation-begin} \rangle$ is found.

→ If redo-only record is found, ignore it.

→ If $\langle T_j, O_j, \text{operation-abort} \rangle$ is found,
skip all log records for T_i until $\langle T_i, O_j, \text{op-begin} \rangle$ is found.

→ Stop scan when $\langle T_i \text{ start} \rangle$ is found.

→ Add $\langle T_i, \text{abort} \rangle$ to log.

Eg: $\langle T_0 \text{ start} \rangle$
$\langle T_0, B, 2000, 2050 \rangle$
$\langle T_1 \text{ start} \rangle$
$\langle \text{checkpoint } \{T_0, T_1\} \rangle$
$\langle T_1, C, 700, 600 \rangle$
$\langle T_1 \text{ commit} \rangle$
$\langle T_2 \text{ start} \rangle$
$\langle T_2, A, 500, 400 \rangle$
$\langle T_0, B, 2000 \rangle$
$\langle T_0 \text{ abort} \rangle$

$\langle T_2, A, 500 \rangle$
$\langle T_2 \text{ abort} \rangle$

→ T_1 has committed
→ T_0 has been rolled back completely before system crash
→ Value of B is restored during T_0 rollback.
→ Checkpoint have active transaction T_0 and T_1.

In redo phase

→ Initially undo-list contains {T_0, T_1}

→ It finds ⟨T_1 commit⟩, so T_1 is removed from undo-list. T_1 is redone. ⇒ undo-list = {T_0}

→ It finds ⟨T_2 start⟩, so T_2 is added to undo-list. ⇒ undo-list = {T_0, T_2}

→ It finds ⟨T_0 abort⟩, so T_0 is removed. ⇒ undo-list = {T_2}

In undo phase

→ It scans log backward from the end.

→ When it finds T_2 updating A, the old value of A is resored and redo-only log record writted to log.

→ It finds start record of T_2, then abort T_2 is added to log.

→ Undo-list is now empty, so undo phase terminates.

Chapter-9
Advanced Database System

Distributed Database System

→ The database is stored on several computers known as sites.
→ Each site is loosely coupled.
→ There is no any shared physical component.
→ Communication between sites is done via network.
→ Local transaction access data only from site where it was initiated.
→ Global transaction access data from different sites.

→ Reasons for building distributed database system

a) Users at one site is able to access data residing at other sites.

b) Each site is able to retain a degree of control over data that are stored locally. It has global administrators and local administrators.

c) Even if one site fails, the other sites are functional. With data replication on several sites, failure of a site does not affect the system.

→ Disadvantages

a) High complexity for coordination among sites.
b) Expensive software development cost.
c) High potential for bugs.
d) Increase processing overhead.

→ Homogeneous Database
- → All sites have identical DBMS software.
- → Agree to cooperate in processing user requests.
- → Each site surrender part of autonomy as a right to change schema.
- → It appears as a single system for users.

→ Heterogeneous Database
- → Each site uses different schema.
- → Difficulty in query and transaction processing.
- → Provide limited facility for cooperation.

→ Data Replication
- → Approach to store distributed data in which system contains multiple copies of data in different sites.
- → Full replication indicates storing at all sites.
- → Fully redundant indicates each site having entire copy of database.
- → It ensures availability, parallelism and reduced data transfer.
- → Update cost is high
- → Difficult for concurrency control.

→ Data Fragmentation
- → Approach to store distributed data in which relation r is divided fragments with sufficient information to reconstruct r.
- → Each schema should have common candidate key to ensure lossless join.
- → In horizontal, each tuple of r is assigned to one or more fragments. It allows parallel processing and easy access.
- → In vertical, schema is split into several smaller schemas.
- • It allows parallel processing on relation.

Parallel Database System

→ It contains multiple processors and multiple disks connected by a fast interconnection network.

→ It improves processing and I/O speeds.

→ Many operations are performed simultaneously.

→ Coarse grain consists few powerful processors.

→ Fine grain consists thousands of smaller processors.

→ Performance is measured using throughput and response time.

→ Running a given ~~time~~ task in less time by increasing parallelism is called speedup.

→ Handling a larger tasks by increasing parallelism is called scaleup.

→ Speedup is linear if equals to N while scaleup is linear if equals to 1.

$$\text{Speedup} = \frac{\text{small system elapsed time}}{\text{large system elapsed time}}$$

$$\text{Scaleup} = \frac{\text{small system small task ET}}{\text{large system large task ET}}$$

→ Shared Memory Architecture

→ Processors and disks share a common memory via bus or network.

→ Provides efficient inter-processor communication.

→ Can not be scaled for more processors.

→ Shared Disk Architecture

→ All processors have access to all disk via network.

→ Each processor has private memory.

→ If one processor fails, others can recover the system.

→ Processor communication becomes slower.

→ Shared Nothing Architecture

→ Each processor has its ~~own~~ memory and disks.

→ Communication between processor is via a network.

→ It minimizes interference of resource sharing.

→ Cost for communication and non-local disk access is high.

→ Hierarchical

→ Combines all the architectures.

Data Warehouse Database

→ Datawarehouse is a single, complete and consistent data store obtained from a variety of sources made available to end users in the understandable form and used as a business context.

→ It is subject oriented, integrated, time-variant and non-volatile collection of data to support decision making process.

→ Rules of Data Warehouse

→ Separated from operational environment.

→ Data is integrated.

→ Contain historical data over a long period of ~~p~~time.

→ Data is subject oriented.

→ Mostly read only with periodic batch updates.

→ Data contains levels of details (current, old, highly summarized)

→ Metadata is critical component.

→ Contains chargeback mechanism for resource usage.

→ Phases of Data Mining

1) Data Preparation (Identify main data sets to be used)
2) Data Analysis and classification (Study of data to identify common patterns)
3) Knowledge Acquisition (Analysis result used with KA algorithms)
4) Prognosis (Predict future behavior)

→ Need for Data Warehouse

→ Huge amt. of operational data to be used for strategic decision making.
→ Provides historical data for analysis.
→ Stores good quality data.

→ Application Areas

→ Online Analytical Processing (OLAP) → analysis of complex data from warehouse.
→ Decision Support System (DSS) → making complex and imp decisions.
→ Data Mining → searching data for unanticipated new knowledge.

→ Warehouse Architecture

1) Warehouse Server (Generally RDBMS)
2) OLAP server
3) Clients (Query tool, analysis tool, data mining tool)

Spatial Database System

→ It is a ~~datab~~database system with additional capabilities for handling spatial data.

→ It provides spatial data types (SDT). → point, line, region

→ It also supports relationships among SDT.

→ Spatial data is the information about physical object that can be represented by numeric values in geographic coordinate system.

→ Application Areas

1) Geographical Information System
2) Environmental System
3) Corporate Decision Support System.
4) Battlefield Soldier Monitoring System

→ Modelling

→ Spatial objects

point : represent object by its location in space
line : represent connection in space
region : represent extent in 2d-space

→ Coverages

partition : set of region objects

→ Networks : graph consisting of set of points and lines objects.

→ Spatial Relationships

topological : adjacent, inside
direction : above, below
Metric : distance

Object Oriented Database System

→ It maintains correspondance between real world and database objects without loosing integrity and identity.

→ Object has state and behavior.

→ Entity is represented as a class.

→ Instances of the class are objects.

→ Within an object, the class attributes takes specific values.

→ Object Relational Mapping

→ ORM is a programming technique for converting data between incompatible type systems into object oriented programming language.

→ This creates a virtual object database, which can be used from within the programming language.

→ It is necessary because objects are almost non-scalar values while the DBMS can only store and manipulate scalar values organized within tables.

→ ORM convert object values into group of simpler values for storage in the database.