Database Management System

FOR DIPLOMA STUDENTS

Notes Prepared by
RABI KUAMR DARJI
IT Dept.

JHARSUGUDA ENGINEERING SCHOOL, JHARSUGUDA

JHARSUGUDA ENGINEERING SCHOOL
ESTABLISHED - 1956
A GOVERNMENT OF ODISHA POLYTECHNIC

CHAPTER-01
Basic Concepts of DBMS

Data :- Any raw facts & figure, unorganised, it can be anything. Ex Name, Number.

Information :- The processed data that is given meaning by its contexts is calles Information.

What is database ?

→ A Database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people your know. This collection of related data with an implicit meaning is a data base.

What is DBMS ?

→ A data base Management System (DBMS) is a collection of programs that enables users to create and maintain a data base. The DBMS is a general purpose software system that facilitates the processes of defining, constructing, manipulating and sharing data base among various users and applications.

* Purpose of Database System

(i) Data Redundancy

⇒ Repeatation of data

→ In file processing System, The Same data may be duplicated in several files.

(ii) Data Inconsistency

→ File System approach can also result in data inconsistency

→ Inconsistency means that files may contain different data of the same student.

(iii) Difficult in accessing data

→ Conventional file processing environments do not allow needed data to be retrived in a convenient and efficient manner.

(iv) Data Isolation

→ Data is scattered in various files, and files may be in different formats. It is difficult to write new application program's to retrieve appropriate data.

(v) Concurrent access

→ In order to improve the overall performance of the System and obtain's faster responses the many Systems allow multiples users to update the data simultaneously. In such environment interaction of concurrent update may result in inconsistent data.

(vi) Security Problem

→ Not every user of the database System should be able to access all the data. ~~For~~ eg In a banking System, Pay roll personal need only see that part of the database that has information about and the customer accounts. Since application programs added to the System in an ad-hoc manner, it is difficult to enforce such security constraints.

(vii) Integrity Problems

→ Integrity constraints are a set of rules. It is used to maintain the quality of information.

→ Integrity Constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.

→ Thus, Integrity problem constraint is used to guard against accidental damage to the database.

(viii) Data Atomicity

→ Data atomicity means that either a transaction should take place as a whole or it should not take place at all.

→ It ensures that the database will always have correct and consistent data.

→ A collection of all steps to complete process is known as transaction.

Explain Data abstraction :

→ Data abstraction means hide certain details of how the data are stored and maintained.

→ Data abstraction is the ability of the database system to provide an abstract view of the data and hide certain details of how the data are stored and maintained.

→ There are three levels of data abstractions

(i) Physical level :- / Internal level

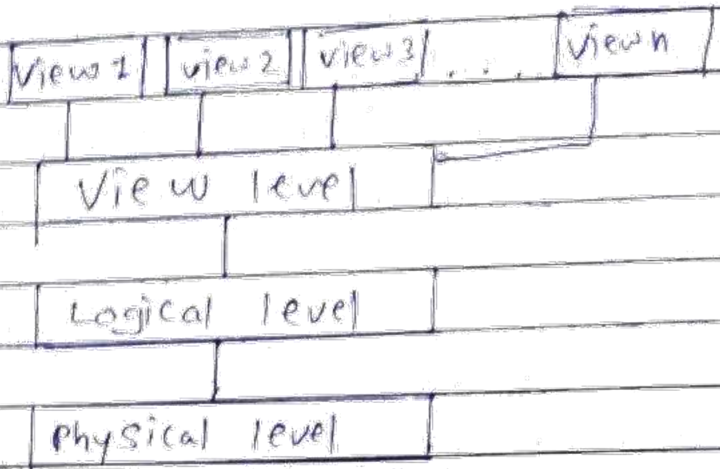
It is the lowest level of abstraction that describes how the data are actually stored. The physical level describes complex low level data.

(ii) Logical level :- / Conceptual level

It is the middle level of abstraction that describes what data are stored in the database and what relationship exist among those data.

(iii) View level :- / External level

It is the highest level of abstraction that describes only part of the entire database



[The three levels of data abstraction]

* Data base Users

Database Users are the one who really use and take the benefits of database.

(i) Native users :-

Who don't have any DBMS knowledge but they frequently use the database applications in their daily life to get the desired results.

Example:-

Online library System, ticket Booking Systems, ATM, etc. which has existing application and users use them to interact with the database to fulfill their request.

(ii) Application programmers :-

They are the developers who interact with the database by means of DML queries. These DML queries are written in the application programs like C, C++, JAVA, pascal etc. These queries are converted into object code to communicate with the database.

Example :- Write a C program to generate the report of employees who are working in particular department will involve a query to fetch the data from database. It will include a embedded SQL query in the C program.

(iii) Sophisticated Users :-

Sophisticated Users have great knowledge of query language so they use database query language to acess information from the database to meet their complicated requirement.

Example :-

Users Such as a business analyst, scientists etc. interact with the Database without writing any application programs.

(iv) Specialized Users :-

They are also Sophisticated Users who write specialize database applications that do not fit into the traditional data processing framework. They are the developer who develop the complex programs to the requirement.

Example :-

Computer Aided Design (CAD) system, Expert Systems knowledge Based System, etc. That Store Complex data types (graphics and audio data) & environment modelling systems.

(V) Online Users / Stand - alone Users :-

These Users will have Stand-alone database for their personal use. These kinds of database will have readymade database packages which will have menus and graphical interfaces.

* Data Defination language (DDL)

→ DDL is use for defining and modifying the data & the structure.

(i) Create :-

→ It is used to create a new database table index or stored procedure, or to create object & create new table.

Syntax :- Create ~~table~~ Database Database name;

Create table -

Eg Syntax - Create table <table name>

```
(
    id int (size),
    name varcher ~~char~~ (size),
    designation ~~char~~ varcher (size)
);
```

(ii) Drop :-

→ Drop command completely removes a table from the database. ~~It can also be used on Databases,~~ to this command will also destroy the table structure and the data stored in it.

Syntax :- Drop Database Database name,

Syntax :- Drop Table Table name

(iii) Truncate :-

→ Truncate command removes all the records from a table. But this command will not destorey the table's structure.

Syntax :-

Truncate table table name;

(iv) ~~Drop~~ Rename :-

→ Rename commands is used to set a new name to any existing table.

Syntax: Rename table old table name to new table name; / alter table student rename to stud;

(v) Alter :-

→ Alter command is used for altering the table structure. or It is used to modify existing database data structure.

- to add a column ~~to~~
- to drop a column from the table
- to change datatype of any column or to modify its size
- to rename any existing column

- Alter : Add a new column

→ Using Alter command we can add a column to any existing table.

Syntax:

Alter table table name ADD (column name datatype);

Example:

Alter table ~~name~~ Studen ADD (address ~~char~~ varchar (200));

- Alter: Add multiple new column

→ Using Alter command we can even add multiple new columns to any existing table.

Syntax:

Alter table table name ADD (column name 1 datatype, column name 2 datatype, Column name 3 datatype);

Example:

Alter table Student ADD (Father name ~~char~~ varchar (50), mother name ~~char~~ varchar (50), dob ~~Date~~ Date);

- Alter: Rename a column

⇒ using Alter command rename an existing table column.

Syntax:	Example:
Alter table table name Rename old column name to new column name ;	Alter table Student Rename address to location ;

- ~~Drop~~ Alter: Drop a column

⇒ Alter command can also be used to Drop or remove columns.

Syntax:	Example:
Alter table table name Drop(column name);	Alter table 'Student' Drop (Address);

- Alter: Add column with default value

⇒ Alter command can add a new column to an existing table with a default value too. The default value is used when no value is inserted in the column.

Syntax:	Example
Alter Table table name add (column name1 datatype1 Default some value);	Alter table Student add ('dob' ~~Data type~~ Date Default '01-Jan-99');

- Alter: Modify an existing column

⇒ Alter command can also be used to modify data type of any existing column.

Syntax:	example:
Alter table table name modify (column name datatype)	Alter table Student Modify (address ~~char~~ varchar (300));

error Column to be modified must be empty to change datatype

- Alter: Rename a column

⇒ Using Alter command rename an existing table column.

Syntax:	Example:
Alter table table name	Alter table Student Rename
Rename old column name	address to location;
to new column name;	

- ~~Drop~~ Alter: Drop a column

⇒ Alter command can also be used to Drop or remove columns.

Syntax:	Example:
Alter table table name	Alter table Student
Drop(column name);	Drop (Address);

- Alter: Add column with default value

⇒ Alter command can add a new column to an existing table with a default value too. The default value is used when no value is inserted in the column.

Syntax:	Example
Alter table table name ADD	Alter table Student ADD
(column name 1 datatype 1	(dob ~~Data~~ ~~type~~ Date
Default some value);	Default '01-Jan-99');

- Alter: Modify an existing column

⇒ Alter command can also be used to modify data type of any existing column.

Syntax:	Example:
Alter table table name modify	Alter table Student Modify
(column name datatype)	(address ~~char~~ varchar (300));

error column to be modified must be empty to change datatype

- To view the change structure of table we describe command.

Syntax: Describe table <table name>.
or Desc table <table name>.

* Data Dictionary

→ A structured place to keep details of the contents a data flow process & data stores.

→ It is a structured repository of data about data.

→ It is a set of definations of all dfd (data flow diagram) elements.

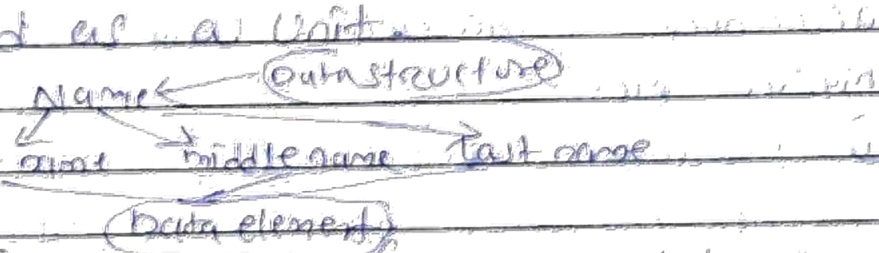
Items to be defined in data dictionary

① Data element: It is a smallest unit of data that provides function and it will not further divided.

Example: data consist of day, month & year.

② Data Structure: It is group of data element handled as a unit.

Example:



③ Data flow: Data flow are data structure is motion.

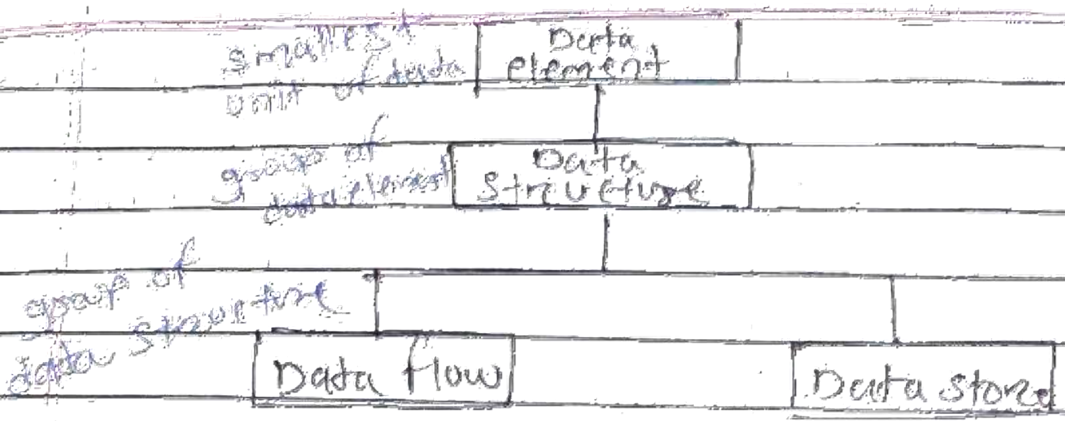
④ Data Stores: Data store are data structure of res

Advantages of Data Dictionary

→ Documentations:- It is valuable reference in any organisation.

→ It improves user communication.

→ Data dictionary is important to build a database.



* Data base Administrador (DBA) :-

⇒ Data Base Administrator is a person or group of person who are responsible for managing all the activities related to database system.

Responsibility of DBA

① Software installation and Maintenance

It is the responsibility of DBA to installed the database software and configure the software according to the need. Many software like oracle, mysql etc.

② Database Accessibility

He decides the user of database and also decides which data can be acess by which user.

③ Validation checks on data

DBA decides which type of data can store in the database or which type cannot. So he put validation checks on data to make it more accurate.

④ Decide the Hardware device

Depending upon the efficiency, performance and cost of the H/w, it is DBA who have the duty of deciding which H/w will suit the company requirement

(5) Decides Data Recovery and Backup method.

It is the DBA who take backup of database in regular time interval. DBA has to decide that how much data should be backup. Recovery of database is done by DBA if they have lost the database or the damage.

(6) Database Design:

The logical design of the database is designed by the DBA.

DBA also design physical design, integrity control, external model design.

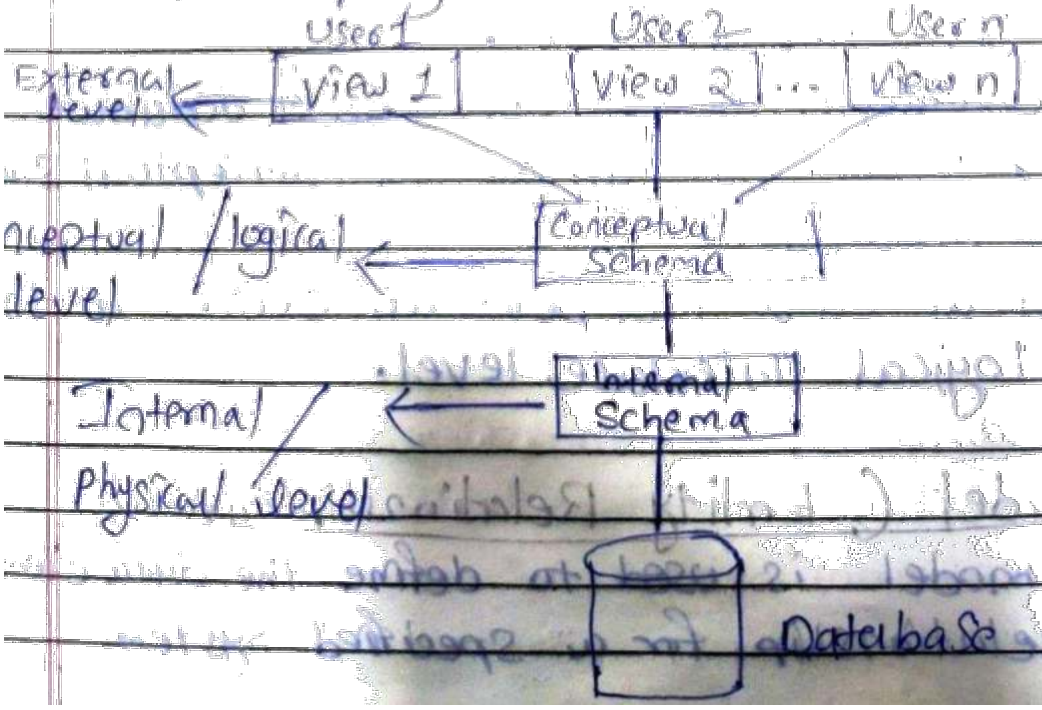
CHAPTER-02

DATA MODELS

* Data Independence:-

→ The ability to modify a schema definition in one level without affecting a schema definition in the next higher level is called data independence.

→ Data Independence helps us to keep data separater from all programs that make use of it.



Types of Data Independence
(i) Logical Data Independence
(ii) Physical Data Independence

(i) Logical Data Independence:-

→ Chenging the logical Schema (Conceptual level) without changing the external Schema (view level) is called logical data independence.

→ Logical data independence is used to separate the external level from the conceptual level.

→ ~~If we make any change~~

→ If we make any change at the conceptual level of data, then it does not affect the view level.

→ Logical data independence occurs at the user Interface level.

(ii) Physical Data Independence:-

→ Making changes in physical Scehema without changing the logical Schema is called physical Data independence.

→ It is used to separate the conceptual level from the Internal level.

→ If we change the Storage size of database system Server, it will not affect the conceptual structure of the database.

→ physical data independence occurs at the logical Interface level.

* ER Model (Entity Relationship model)

→ This model is used to define the data elements and relationship for a specified system.

→ It is the most popular conceptual model or object based model used for designing a database.

→ ER model views the real world as a set of basic objects (entities), their attributes & relationship among these objects (entities).

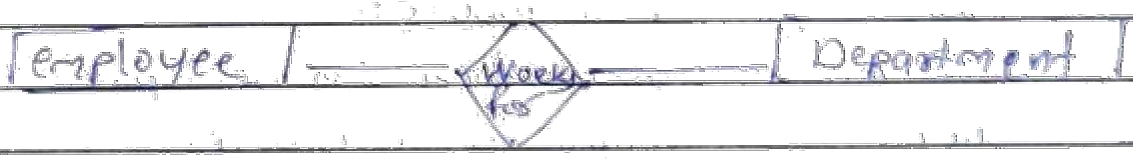
→ Entities, attributes & relationships are the basic construct of an ER model.

→ ER data model describe the structure of a database with the help of diagram called as ER diagram.

- Components of ER Diagram :-

(1) Entity:- An entity is an object or distinguisable thing in the real world.

Ex - Car, Students, product, employee etc.



Entity types:- A set or a collection of entities that share the same attributes but different values is known as an entity type.

Student → (Entity type)

ibute
mn

ID	Name	Age
e1	Ram	14
e2	Shyam	17
⋮	⋮	⋮
en	Mohan	15

→ Entity

Entity Sets:- It is collection of set of similar type of entity which has same properties.

Domain:- Domain is a set of permitted values for an attribute.

$5 < Age < 100$

(2) Attribute :- ⇒ Each entity has certain characteristics known as attributes.

⇒ The attributes is used to describes the property of an entity.

Ex:- ~~(eg)~~ Student has attributes
(Name, age, ID, weight etc.)

Types of attributes :-

(i) Simple attributes :-

⇒ An attributes which cannot be further subdivided into components is a simple attribute.

⇒ The attributes which cannot be partitioned into smaller sub-part is called simple attributes.

ex:- Student roll number, employee ID, year, price

(ii) Composite attribute :-

⇒ An attribute which can be splitted into components is a composite attribute.

⇒ A Composite attributes ~~which further~~ can be subdivided into smaller subparts which further form attributes.

Ex:- Name – First name, middle name, Last name

address – House no., Street, City, State, Pincode

(iii) Single valued attribute :-

⇒ The attribute which takes up only a single value for each entity instance is single valued attribute.

⇒ The attribute that can have only one value for a given entity are called single valued attributes. ex → age

Ex:- Book title is a single valued attribute as one book can have only one title, The age of a student

(iv) Multi valued attributes:-

→ The attribute which takes up more than a single value for each entity instance is multi valued attribute.

→ The attribute that can have multiple values for a given entity are called multi valued attributes.

Ex:- (Email-id), (phone-no).

(v) Stored attributes:-

→ An attribute that is stored in a database is called stored attribute.

→ Most of the attribute are stored attribute.

Ex:- Student Name, Age, (DOB) Roll no, phone no,

(vi) Derived attribute:-

→ An attribute that is not stored in database but derived from another value is called derived attribute.

→ A derived attribute calculate its value from another attribute.

Ex:- The value of the attribute [Age] can be ~~dexa~~ determined from the current date ~~a~~ of the value of DOB attribute.

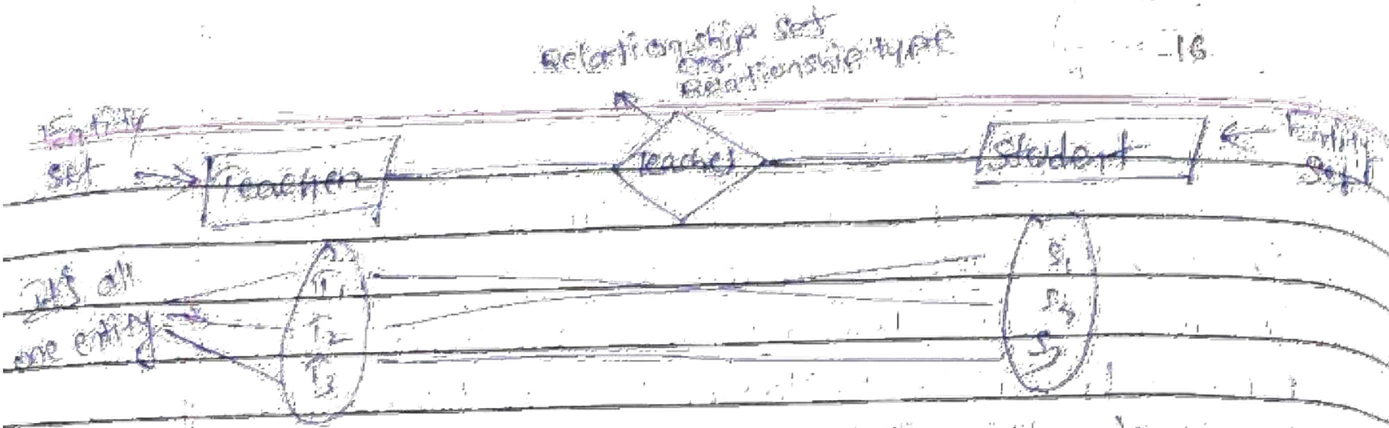
Age - derived attribute.

DOB - stored attribute.

(3) Relationship:-

→ It is an association between two or more entities of same or different entity set.

→ No representation in ER diagram as it is an instance or data.



→ In Relational model relationship is represented either using row in a table.

* Relationship Set / type

→ A set of similar type of relationship.

→ A relationship is defined as an Association among several Entities.

→ Relationships are represented by diamond symbol

→ In relational model we have create new table or by separate column to represent relationship set.

→ Every relationship type has ~~two~~ three components :-

(i) Relationship type name

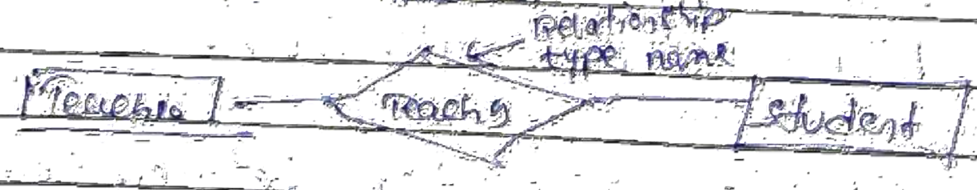
(2) Degree

(3) Cardinality ratio / ~~Participation~~ Mapping Constraints

(1) Relationship type name -

Each relationship type needs to have a unique name to avoid confusion.

Ex:



(2) Degree -

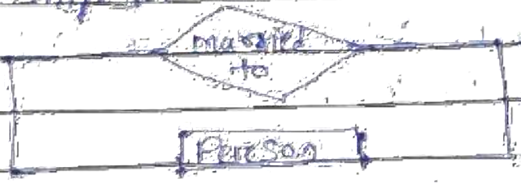
→ It means number of entity set association in relationship set.

→ The degree of a relationship type is the number of participating entity types.

Types of Relationship sets:-

(i) Unary relationship set - Unary relationship set is a relationship set where only one entity set participates in a relationship set.

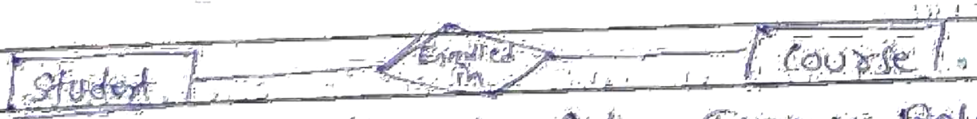
Ex:-



[one person is married to only one person]

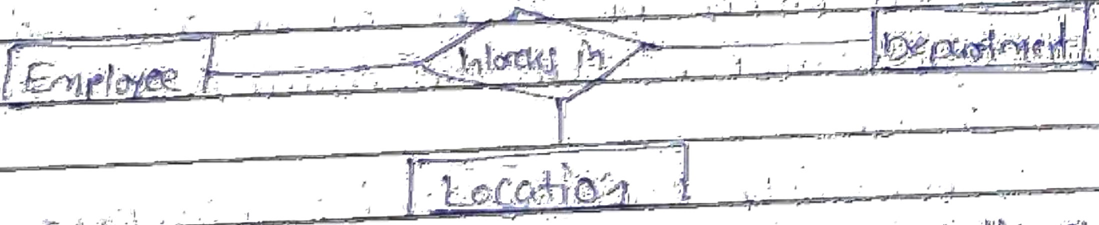
(ii) Binary Relationship set - Binary relationship set is a relationship set where two entity sets participate in a relationship set.

Ex:- [Student is enrolled in a course]



iii) Terenary relationship set - Terenary Relationship set is a relationship set where three entity set parti-cipate in a relationship set.

Ex:-



) N-ary Relationship set - N-ary relationship set is a relationship set where 'n' entity sets participate in a relationship set.

3.) Cardinality Ratio - / Mapping Constraints

Cardinality of relationship type explains maximum number of relationship instances an entity can participate in.

Cardinality constraint defines the maximum number of relationship instances in which an entity can participate.

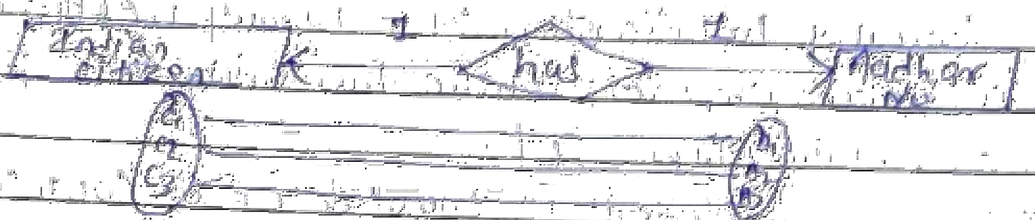
Types of Cardinality ratios-

(i) One to One Cardinality -

→ An entity in Set A can be associated with at most one entity in Set B.

→ An entity in Set B can be associated with at most one entity in Set A.

Ex:-



Here,

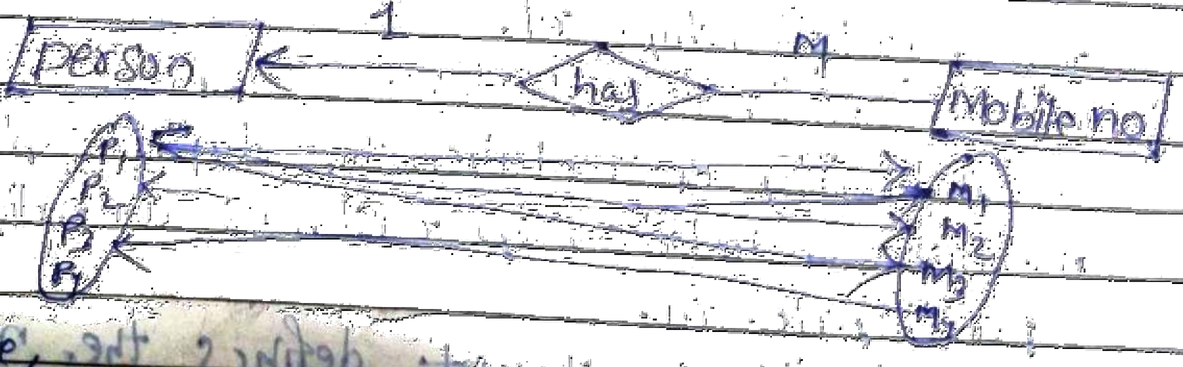
- A man can have at most one Aadhar number.
- A single Aadhar number can be owned by at most one person.

(ii) One to Many Cardinality -

→ An entity in Set A can be associated with any number (zero or more) of entities in Set B.

→ An entity in Set B can be associated with at most one entity in set A.

Ex:-



Here,

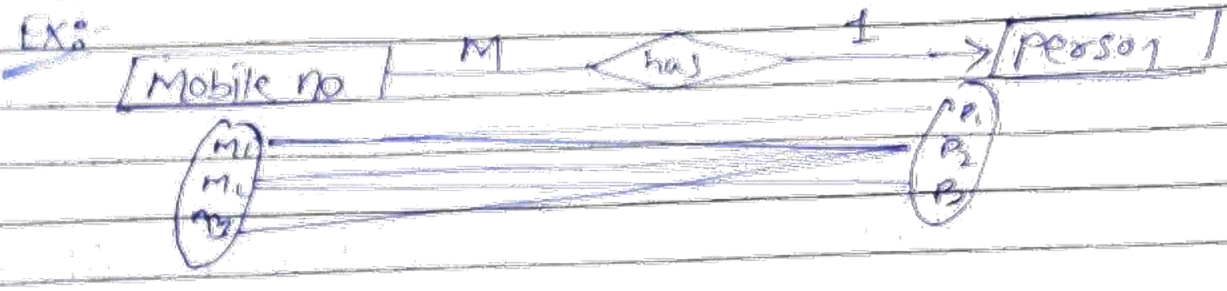
- A man can have more than one mobile number
- A mobile number can be owned by at most one person.

(iii) Many to One Cardinality -

→ An entity in set A can be associated with at most one entity in set B.

→ An entity in set B can be associated with any number (zero or more) of entities in set A.

Ex:



Here,

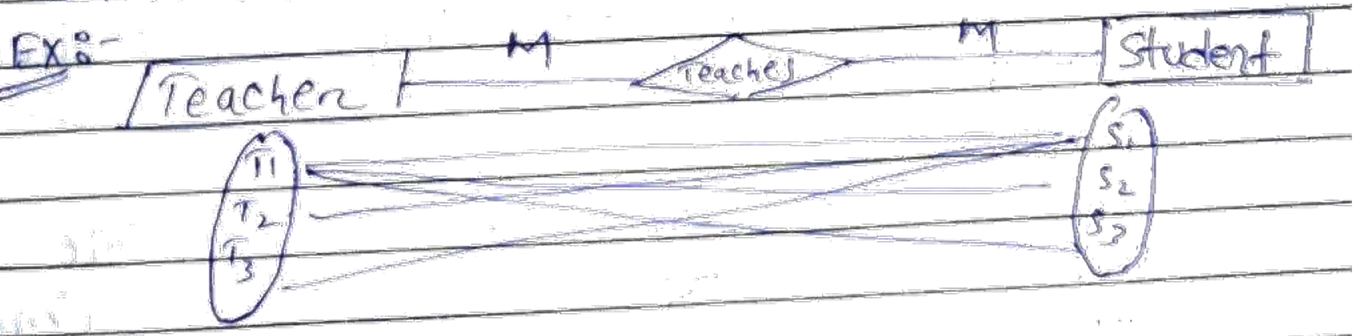
- A mobile number can be owned by at most one person.
- A man can have more than one mobile number.

(iv) Many to Many Cardinality -

→ An entity in set A can be associated with any number (zero or more) of entities in set B.

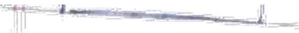
→ An entity in set B can be associated with any number (zero or more) of entities in set A.

Ex:-



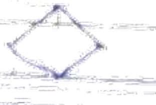
Here,

- A teacher can teach more than one student
- A student ~~can~~ can read from more than one teacher.

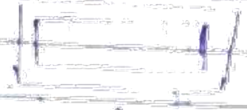


→ (Entity set)

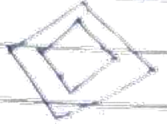
→ (Attribute)



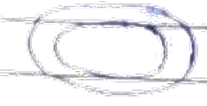
→ (Relationship)



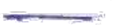
→ (Weak entity)

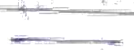


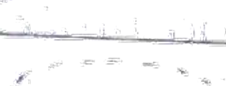
→ (Weak entity / Identifing
Relationship / relationship set)



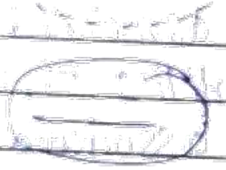
→ (Multivalued Attribute)

(Single line)  → (Partial Participation)

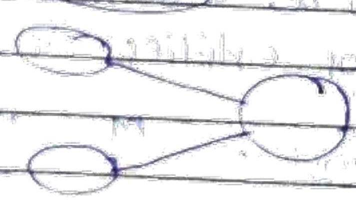
(Double line)  → (Total participation of entity
is relationship set.)



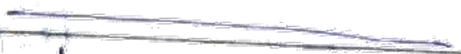
→ (Derived Attribute)



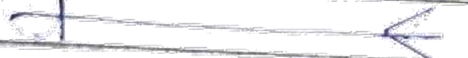
→ (Key Attribute)



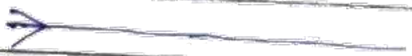
→ (Composite Attribute)



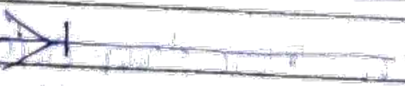
⇒ One to one



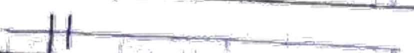
⇒ One to many (mandortory)



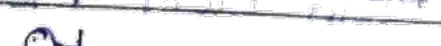
⇒ Many



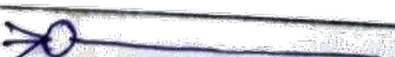
⇒ One or ~~more~~ many (mandatory)



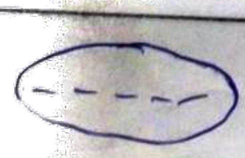
⇒ One and only one (mandatory)



⇒ Zero or one (mandatory)



⇒ Zero or many (optional)

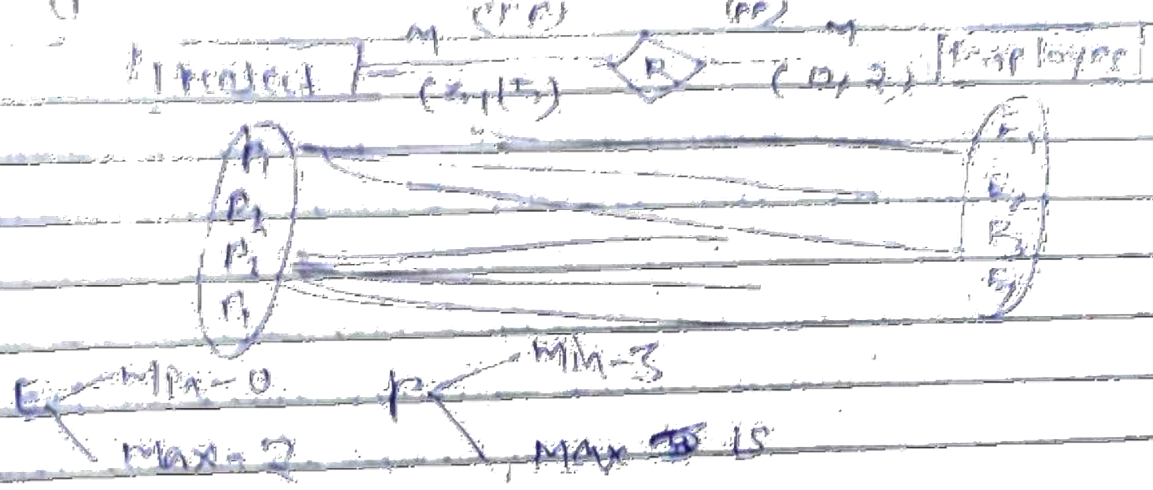


⇒ Discriminator / partial key

* Participation Constraints :-

-> Specifies whether the existence of an entity depends on its being related to another entity via a relationship type.

These constraints specify the minimum and maximum number of relationship instances that each entity can must participate in.



~~Min~~

Maximum cardinality :-

-> It defines the maximum number of times an entity occurrence participating in a relationship.

Minimum Cardinality :-

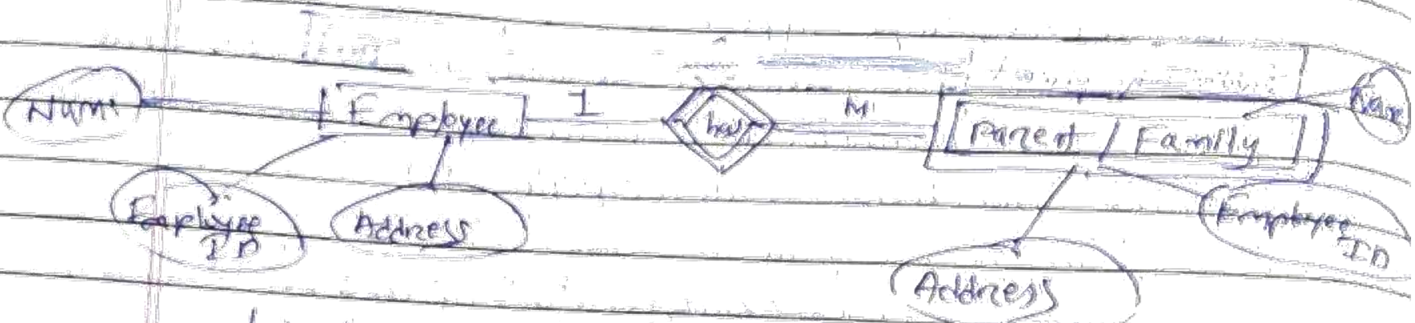
-> It defines the minimum number of times an entity occurrence participating in a relationship.

Partial participation -
If minimum cardinality is 0.

Total paraticipation -
If maximum cardinality is ~~[illegible]~~1. >=

* Weak entity :- An entity that is existence dependent on some other entity is called weak entity type.

Strong entity :- An entity set on which weake entity set depends is called strong entity set.



Weake entity, Set 'Parent or Family' which depends on strong entity set 'Employee'.

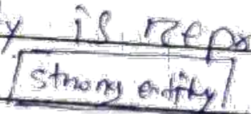
Strong Entity (Strong Entity Set)

-> The Strong entity always have a primary key.

-> Its existence is not dependent on any other entity i.e. it is independent of other entity.

-> A set of strong entities is known as strong entity set.

-> Strong Entity is represented by a single rectangle



Weake Entity (Weak entity Set)

-> The weake entity does not have a Sufficient attributes to form a primary key i.e., Weak entity do not have a primary key.

-> A Weake entity is dependent on a strong entity to ensure the its existence.

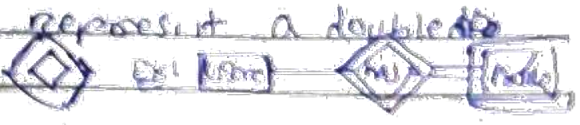
-> A Set of Weak-entities is know as weake entity set.

-> Weake Entity is represented by double rectangle.

Weak entity

→ The existence of a weak entity set depends on the existence of a strong entity ~~of~~ set is called identifying entity set.

→ The relationship associating the weake entity set with the strong (or identifying) entity set is called identifying Relationship.

- Identifying relationship represent a double diamund 

Discriminator (or partial key)

→ ~~AP~~ Partial key of a weake entity set of attribute that distinguishes among all the entities of a weak entity set.

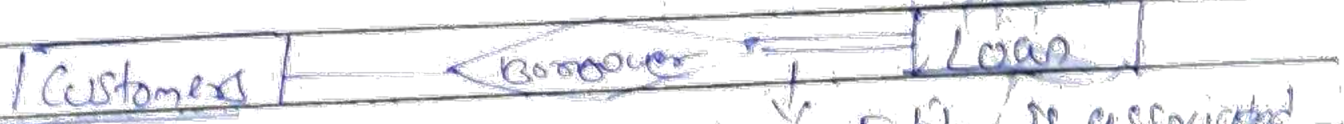
- Discriminator of a weak entity set is underline with a dashed line ....(-----)

Primary key

→ By the help of key attribute we can find a unique row in a table.

Total participation: (Participation Constraints)

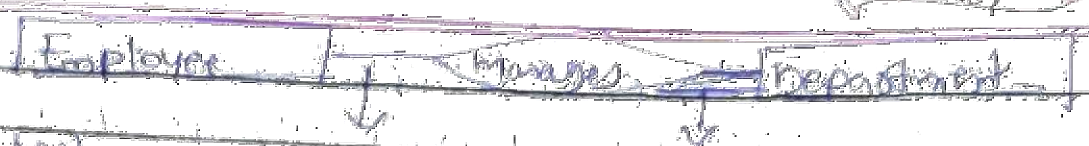
Each entity in the entity set occurs in at least one Relationship in that relationship set.



Each Loan Entity is associated with at least one associated customer.

Partial participation:-

Each Entity in the entity set may not occurs in at least one ~~g~~relationship in that relationship set.



Not every Employee manage a Department

Every Department is managed by at least one Employee (called manager)

* ER Diagram :-

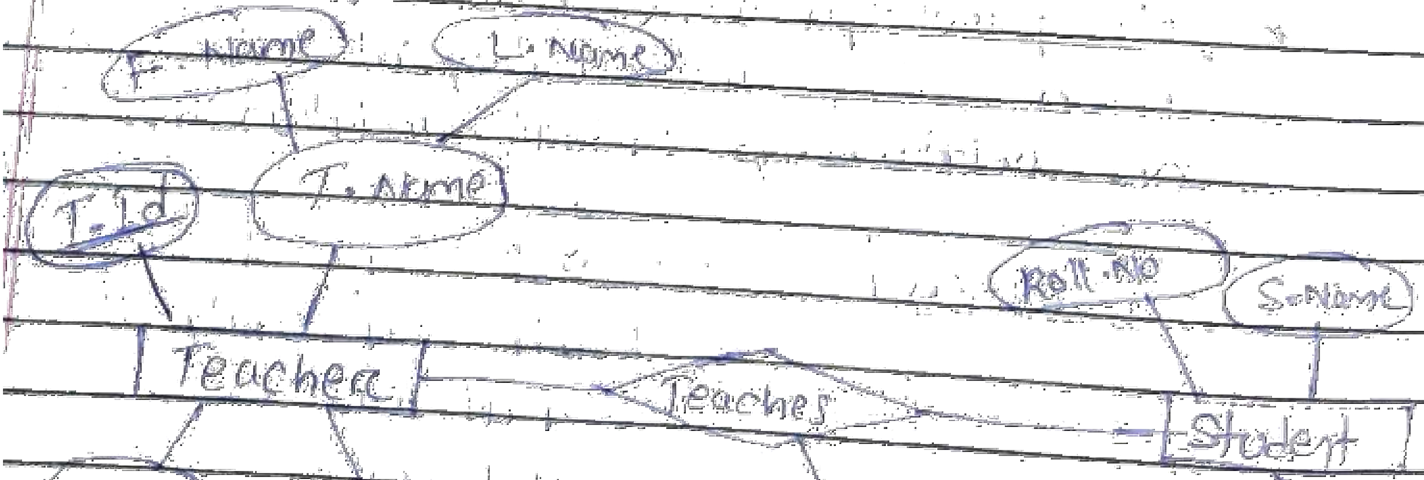
→ It was Introduced by Dr. peter chen in 1976.

→ A nontechnical designed method works on conceptual level based on the perception of real world.

→ Consists of collection of book objects called entities and of relationship among these object and attributes which defines their properties.

→ free from ambiguteen and provides a standard and logical way of visulizing data

→ basically it is a diagrammatic representation easy to understand even by nontechnical users.

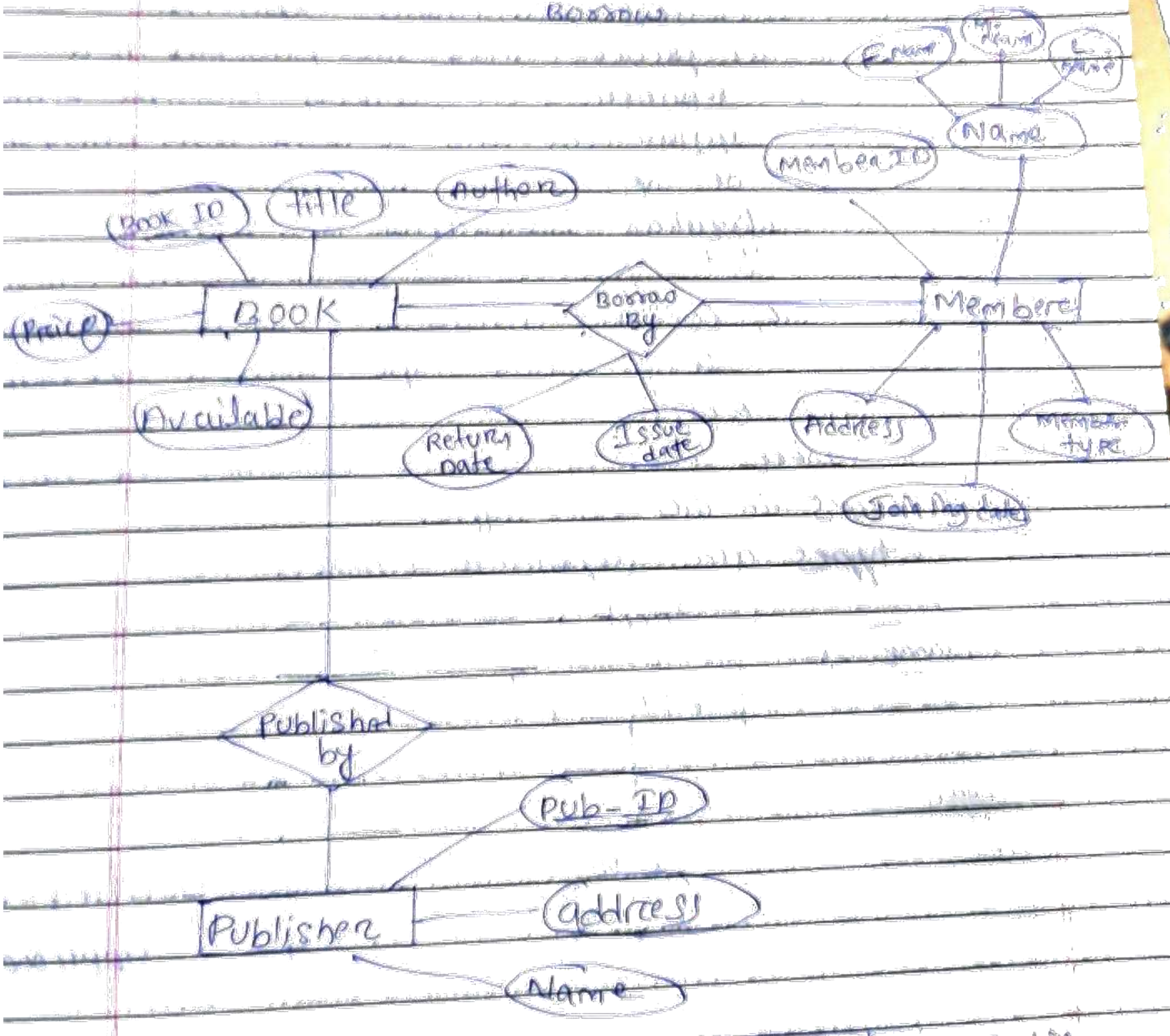


~~ER~~ diagram of (Liberary Management systems)

Entity ⇒ Book, Publisher, Member

Publish

Borrow



□ — Entity

◇ — Relation

⬭ — Attribut

⬭ — Key Attribute

GENERALIZATION (Extendes ER diagram)

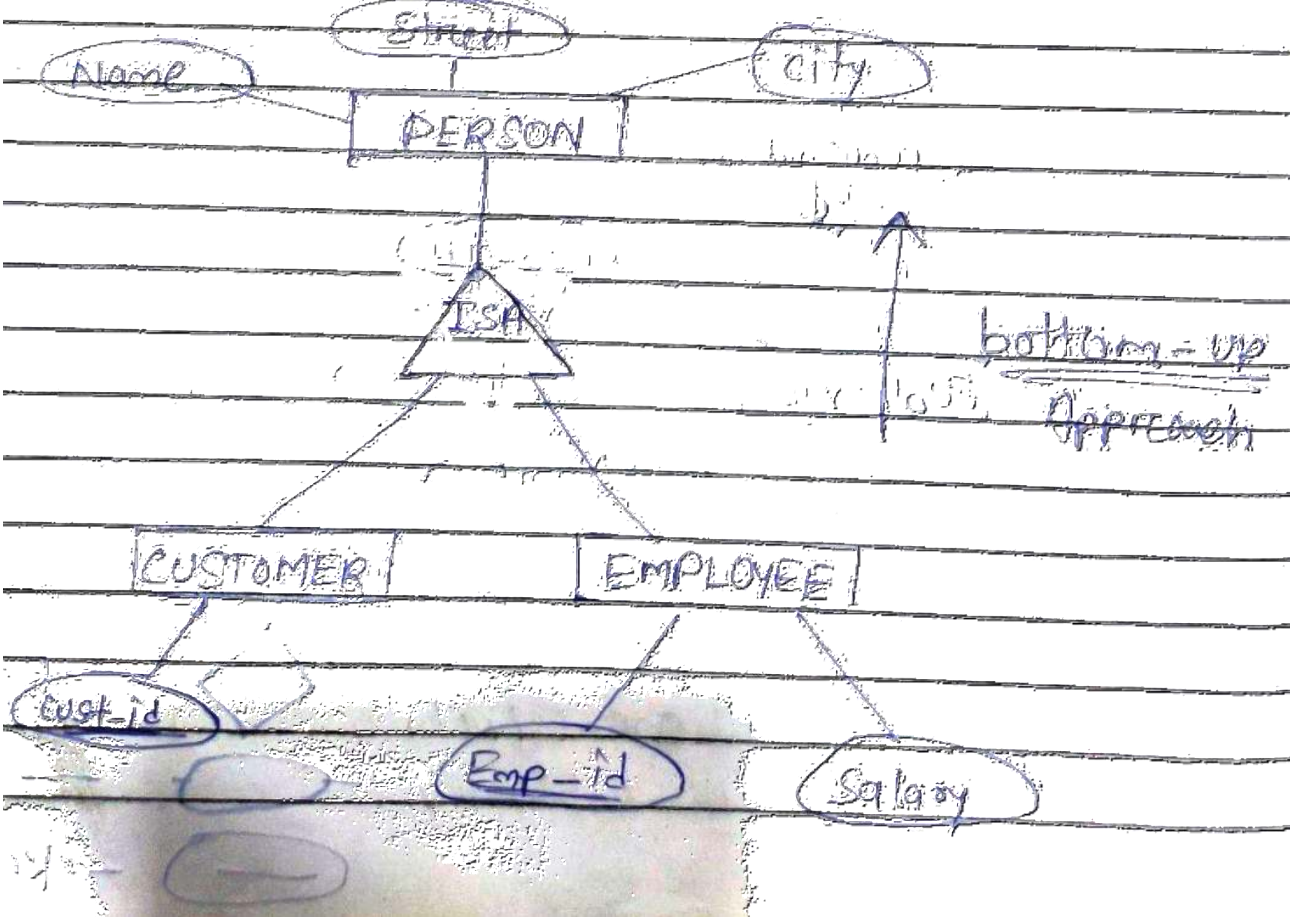
Defination: The refinement from an initial entity set into successive levels of entity sub-groupings represents a top-down design process in which distinctions are made explicit.

→ The design process may also proceed in a bottom-up manner, in which multiple entity set are sythesized into a high level entity set on the basis of the common features.

→ A generalization hierarchy is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher-level entity type called a supertype or generic entity.

→ The lower level of entities becomes the subtype categories to the super type.

→ Subtypes are dependent entities.



SPECIALIZATION (Extended ER diagram)

Definition: An entity set may include subgroupings of entities that are distinct in some way from other entities in the set.

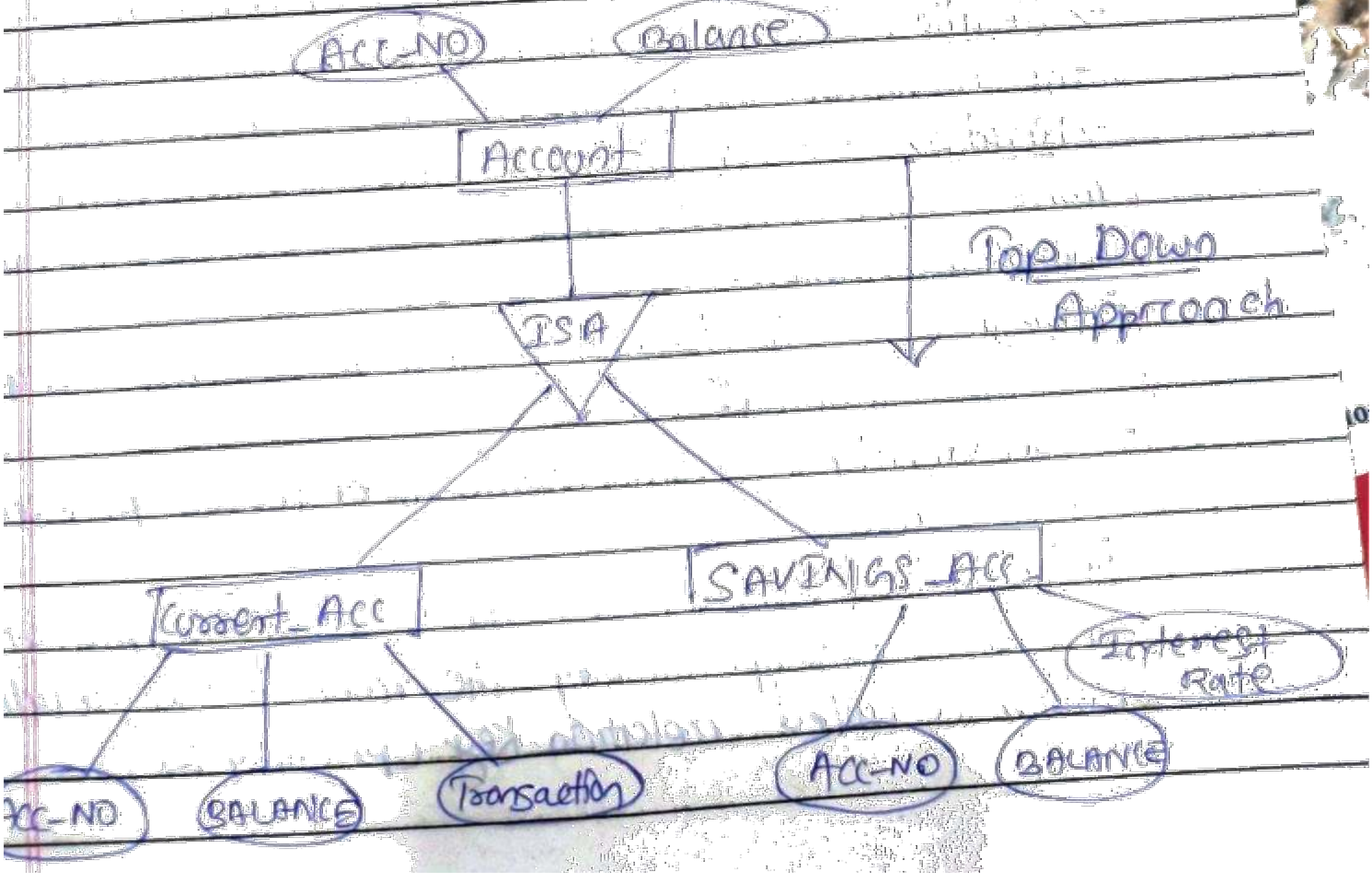
For instance, a subset of entities ~~as~~ within ~~an~~ an entity set have attributes that are not shared by all the entities in the entity set.

The process of designating subgroupings within an entity set is called specialization.

→ Specialization is opposite to Generalization.

→ It is a top down approach.

→ Specialization is the process of defining the subgroups of a given entity type or we can say that in specialization an entity is divided into subentities based on their characteristics.



* Relational Model

Relational model represents how data is stored in Relational Database. A relational database stores data in the form of relations (tables).

→ Relational model can be represented as a table with columns and rows.

- Each row is know as a tuple.
- Each table of the column has a name or attribute.

→ Relation:- A relation is a table with columns and rows.

→ Attribute:- An attribute is a named column of a relation.

→ Domain: A domain is the set of allowable values for one or more attributes.

Students

Roll No	Name	Ph. NO
1	Ajay	9898374332
2	Raj	9874442111
3	Vijay	8923432411
4	Aman	8886462644

→ Tupple: A tuple is a row of a relation.

→ Relation Schema: A relation Schema represents the name of the relation with its attributes.

→ Relation instance (State): Relation instance is a finite set of tuples. Relation instances never have dupplicate tuples.

→ Degree: The total number of columns or attributes in the relation.

→ Cardinality: Total number of rows present in the table.

→ Relation Key: Every row has one or multiple attributes that can uniquely identify the row in the relation, which is called relation key (primary key).

* Hierarchical Model

Hierarchical model, data is organized into a tree like structure with each record is having one parent record and many children.

→ The hierarchical model arranges records in hierarchy like an organizational chart...

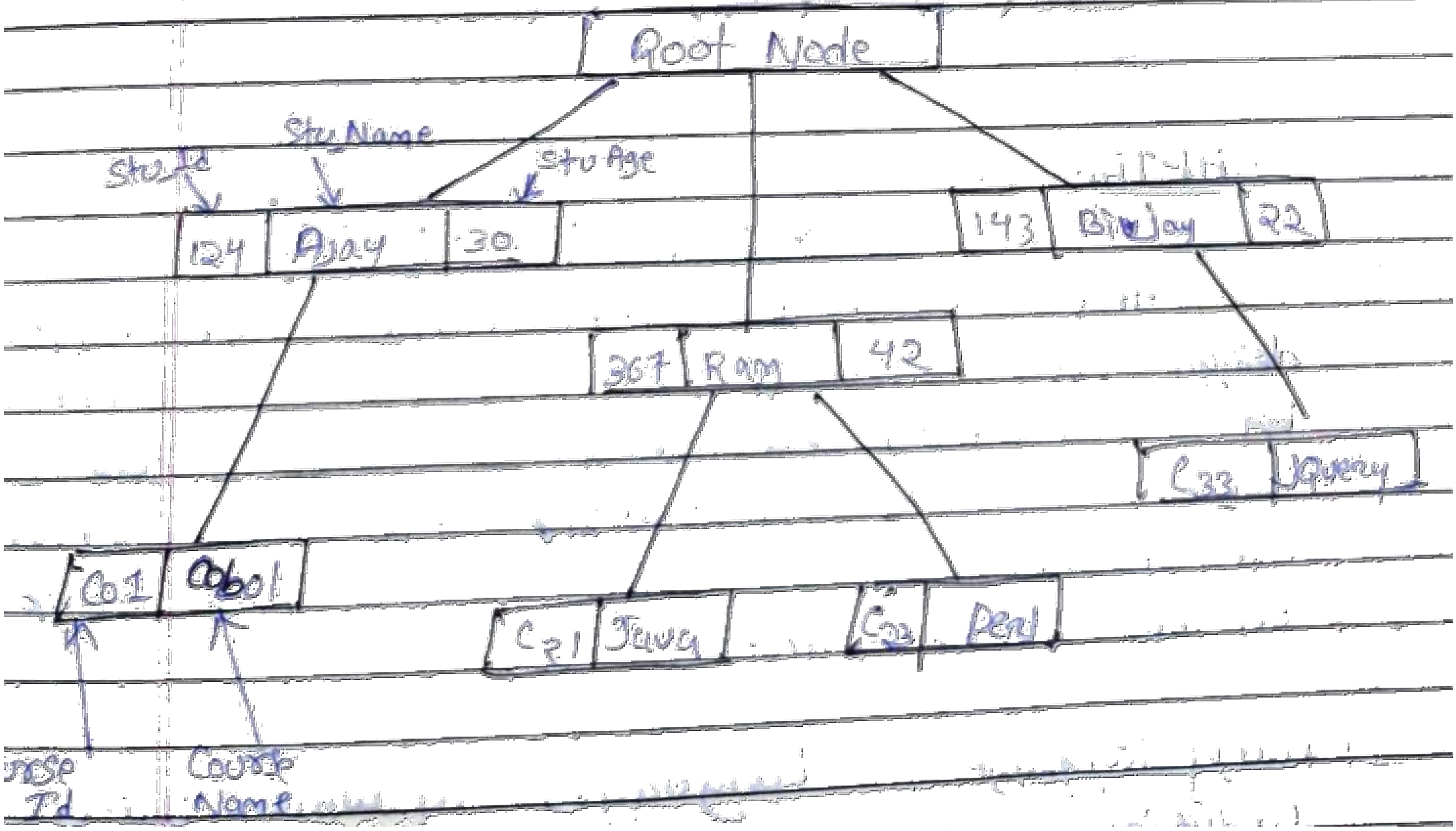
→ Each record type in this model is called a node or segment.

→ A node represents a particular entity.

→ The top most node is called root. Each node is a subordinate of the node that is at the next higher level.

→ A higher level node is called parent and lower level node is called child. A parent node can have one or many child node.

→ A child node can have only one parent node.



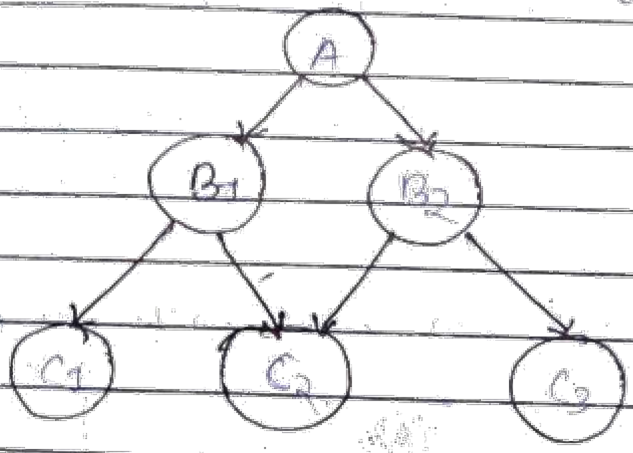
* Network Model

The network model is the extension of the hierarchical structure because it allows many to many relations to be managed in a tree-like structure that allows multiple parents.

→ The difference is that child node can have more than one parent nodes.

→ The child nodes are represented by arrows in network model.

→ It also provides more flexibility than hierarchical model.

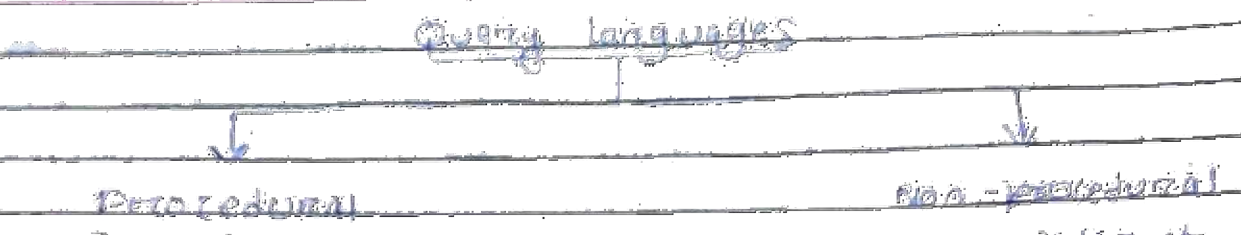


CHAPTER - 03

Relational Database

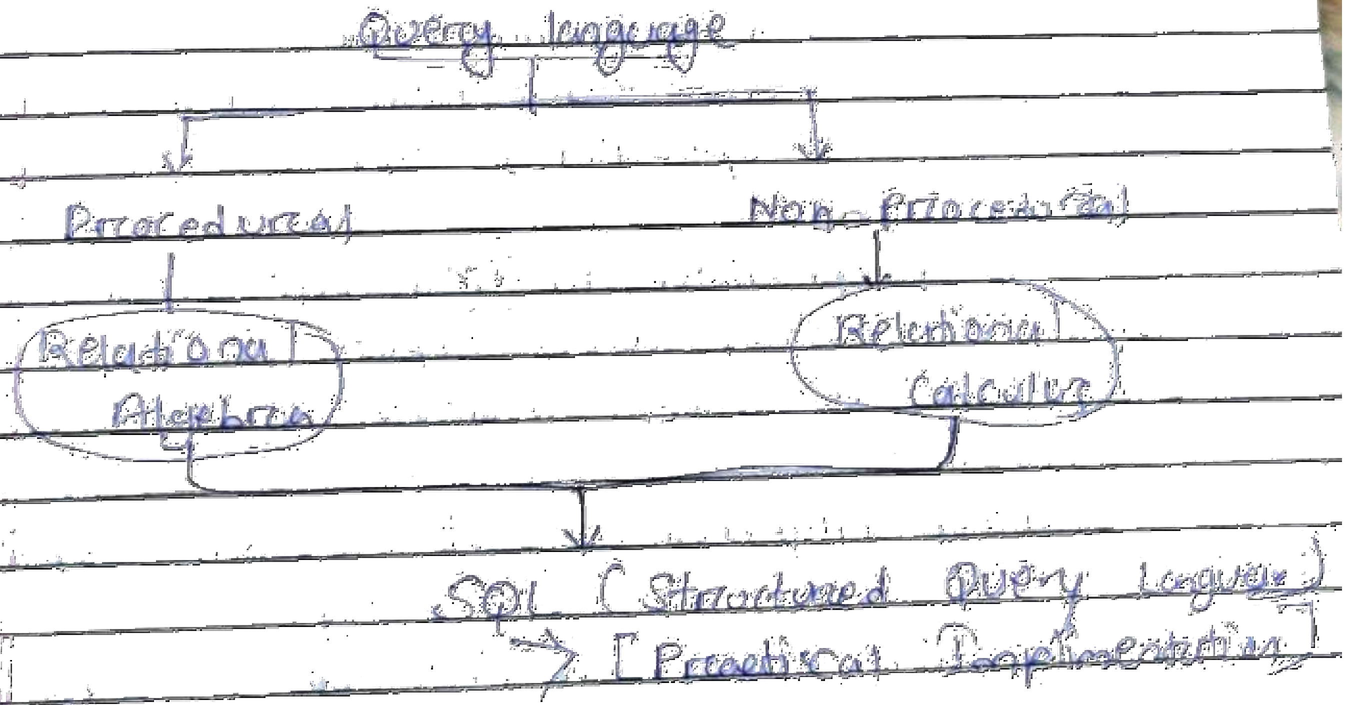
After designing of database i.e ER diagram design then converting it into relational Model followed by normalization and indexing now task is how to store, retrieve and modify data in data base. though here we will be concentrating more on the query part.

Query language - Languages in which user request some Information from the database.



Procedural Query language - Here user interact with the system to perform a sequence of operations in order to produce ~~of~~ ~~described~~ desired result. User tells what data to be retrieved and how to be retrieved.

Non procedural Query language - Here user described the desired information without giving the specific procedure for obtaining the information



In practice we use RDBMS (practical implementation of relational model)

→ SQL is used to write Query on it

→ So Relational model is a conceptual / theoritical frame work and RDBMS is it implementation.

→ Relational Algebra (procedurcal) and Relational calculus (non-procedurcal) are mathematical system or query languages used on Relational model.

Relational Model	R DBMS
RA, RC	SQL
Algorithm	proogram, Code
Conceptual	Reality
Theoritical	Practical

Relational Algebra

→ It is one of the formal query language associated with relational Model.

→ Like Any orderc Mathematical system is defines a number of operators and use relation (table) as operands.

→ Every operator in relational algebra take one or two relation are input argument and generate a single relation as a result without a Name.

→ Relational Algebra do not consider duplicacy as it is based on Set theory.

→ In each query we describes a step by step procedures for computing the desired result so procedurcal QL.

→ No use of English keyword.

Operator

Basic/fountamental	Derived
Select (σ)	Natural Join (⋈)
Proeject (π)	Intersection (∩)
Union (U)	
Set difference (−)	

(1) Select operator (σ)

→ Select is a unary operator, so it can take only one table as input.

→ Select is a fundamental operator.

→ Main idea of select is to find those tupples/ rows in a relation which satisfies a given condition.

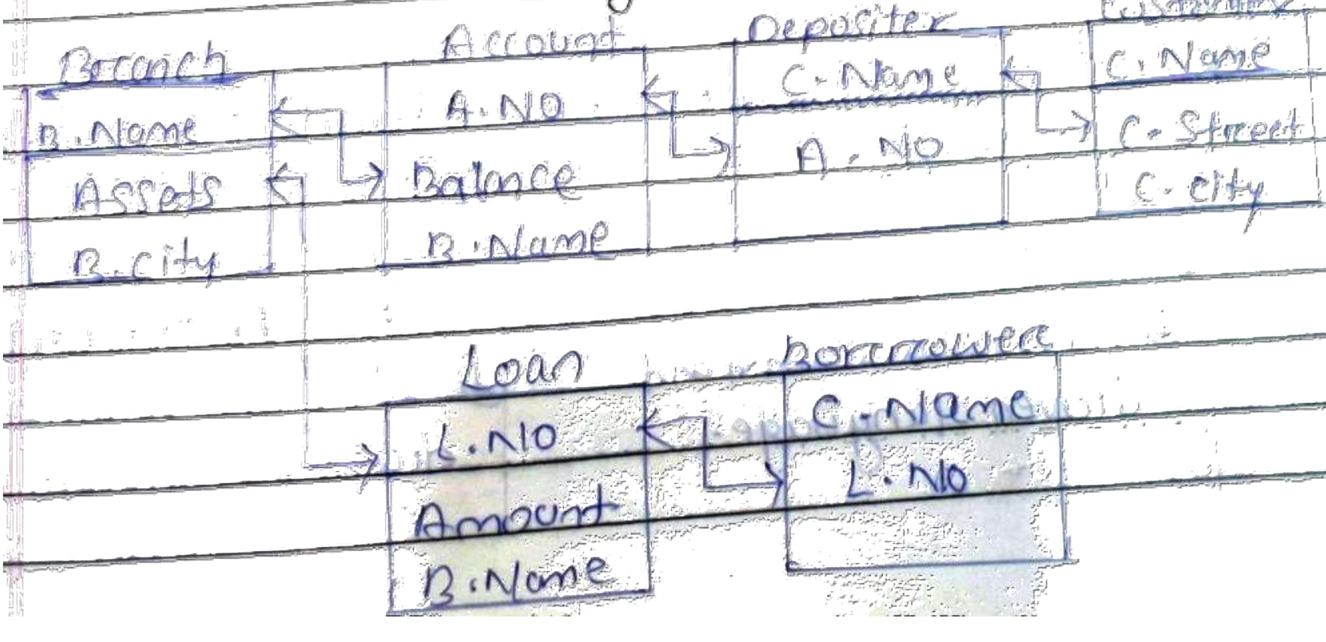
→ Syntax $\sigma_{\text{condition/predicate}}$ (table Name)

→ It has same function as of where clause in SQL.

→ Minimum no. of tupeles selected is 0.

→ Maximum no of tupples selected all the tupples.

Banking Scheema



Q1 Find the details of Accounts balance > 10000

→ $\sigma_{Balance > 1000}(Account)$

Q2 Find the details of the customer who live in delhi?

→ $\sigma_{customer\_city = 'Delhi'}(Customers)$

Q3 Find the details of those loan having Amount <= 5000 and from North Delhi.

→ $\sigma_{Amount \leq 5000}(\sigma_{B.Name = 'N.D'}(Loan))$

OR

$\sigma_{B.Name = 'N.D'}(\sigma_{Amount \leq 5000}(Loan))$

OR

$\sigma_{B.Name = 'N.D' \wedge Amount \leq 5000}(Loan)$

Q4 Find those branch details which are in delhi or having assets more than 10,00,000?

→ $\sigma_{Branch\_city \vee Assets > 10,00,000}(Branch)$

* Difference between Procedural and Non procedural language

Procedural	Non Procedural
① It is a Command driven language.	① It is a function driven language.

(2) Its efficiency is more than than Non-procedural	(2) Its efficiency is less than procedural.
(3) the size of its program is very large	(3) The size of its program is less.
(4) It is not ~~ea~~ suitable for time-critical applications.	(4) It is suitable for time critical applications.
(5) Its semantics are very complex.	(5) Its semantics are very easy.
(6) It only returns restricted data types.	(6) It can return any data types and values.

* Difference between Relational Algebra and Relational Calculus

Relational Algebra	Relational Calculus
(1) It's procedural Query language	(1) It's Non procedural Query language
(2) States how to obtain results.	(2) What result we have to obtain
(3) Relational Algebra is specifies operations order.	(3) Doesn't specify
(4) Domain Independent	(4) Can be dependent
(5) Relational Algebra is programming language	(5) Relational Calculus is a Natural language.

(2) Project (π)

⇒ Unarry operator, take one table at a time

⇒ It is also fundamental operators.

⇒ Main idea of project is select desired columns.

→ Syntax $\pi_{\text{column Name}}(\text{table Name})$

→ It works at select clause of SQL

Q1 Find all branch name of the bank ?

→ $\pi_{\text{branch Name}}(\text{branch})$

Q2 Find all Account No along with there balance ?

→ $\pi_{\text{Account No, Balance}}(\text{Account})$

Q3 Find the Name of all the customers who have loan ?

→ $\pi_{\text{customer Name}}(\text{borrower})$

Q4 Find all the details about branch ?

(branch)

Select and project operation in relational Algebra

Q1 Find the Name of all students from CS branch ?

→ $\pi_{\text{Name}}(\sigma_{\text{Branch}='CS'}(\text{Student}))$

Student

S.ID	Name	Branch
1	A	CS
2	B	ME
3	C	CS
4	D	EE
5	E	CS

Q2 Find those Account No where balance is less than < 1000 ?

→ $\pi_{\text{Account NO}}\left(\sigma_{\text{Balance} < 1000}(\text{Account})\right)$

Q3 Find those Loan Numbers which are from CP branch with amount > 1000 ?

→

$\pi_{\text{Loan NO}}\left(\sigma_{\text{Branch Name} = \text{'CP'} \wedge \text{amount} > 1000}(\text{Loan})\right)$

Q4 Find branch Name and branch city with assets more than > 1,00,000 ?

→ $\pi_{\text{Branch Name, Branch city}}\left(\sigma_{\text{Assets} > 1,00,000}(\text{Branch})\right)$

Union intersection and set difference operator in relational Algebra

Q1 Find the Name of customers who have a loan or an account or both ?

→ $\pi_{\text{Customer Name}}(\text{Depositor})$

$\cup$

$\pi_{\text{Customer Name}}(\text{Borrower})$

Q2 Find the Name of a branch who have accounts but not loan ?

→ $\pi_{\text{branch Name}}(\text{Account}) - \pi_{\text{branch Name}}(\text{Loan})$

Q3 Find the Name of a customer who neither have a loan or an account ?

→ $\pi_{\text{customer Name}}(\text{customer})$

$- \left(\pi_{\text{customer Name}}(\text{Depositor}) \cup \pi_{\text{customer Name}}(\text{Branch}) \right)$

(3) Cartesian Product or Cross product (X)

→ Binary operator, takes two tables at a time

→ Cartesian product is a fundamental operator

→ allow us to combine information between two table

→ $|R_1| = m$, $|R_2| = n$ $|R_1 \times R_2| = m \times n$

Example:-

R_1

A	B
1	a
2	b
3	c

R_2

C	D	F
p	101	y
q	102	z

$R_1 \times R_2$

A	B	C	D	F
1	a	p	101	y
1	a	q	102	z
2	b	p	101	y
2	b	q	102	z
3	c	p	101	y
3	c	q	102	z

OR

$R_1(A, B)$, $R_2(C, D, F)$

$R_1 \times R_2(A, B, C, D, F)$

or

$R_1(1, 2, --- n)$

$R_2(1, 2, ---- m)$

$R_1 \times R_2 = (1, 2 .... m+n)$

A = {1,2} B = {a,b,c}

A×B = {(1,a), (1,b), (1,c)
(2,a), (2,b), (2,c)}

find Customer Name having account balance < 100

Account

A.No	Balance	B.Name
101	50	X
102	70	Y
103	110	Z

Depositer

C.Name	A.No
1	101
2	102
3	103

(Account × Depositer)

Account X Depositer

A.No	Balane	B.Name	C.Nam	Depositer Account No.
101	50	X	1	101
101	50	X	2	102
101	50	X	3	103
102	70	Y	1	101
102	70	Y	2	102
102	70	Y	3	103
103	110	Z	1	101
103	110	Z	2	102
103	110	Z	3	103

(Account × Depositer))

σ Account.A.No = Depositer.A.No

σ balance < 100

Q. Find those account numbers ~~who~~ which are in Delhi?

$$\rightarrow \pi_{\substack{\text{Account} \\ \text{NO}}}\left(\sigma_{\substack{\text{Branch city='delhi'} \\ \cap \\ \text{Branch . Branch Nome = Accout . Branch Name}}}(\text{Branch} \times \text{Account})\right)$$

Q. Find those Customers Name who have a loan from a branch having assets more than 10,00,000.

$$\rightarrow \pi_{\substack{\text{Customer} \\ \text{Name.}}}\left(\sigma_{\substack{\text{Assets} > 10,00,000 \\ \cap \\ \text{Branch . Branch Name = Loan . Branch Name} \\ \cap \\ \text{Loan . Loan NO = Borrower . Loan NO}}}(\text{Branch} \times \text{loan} \times \text{Borrower})\right)$$

CHAPTER - 04

NORMALIZATION IN RELATIONAL SYSTEM

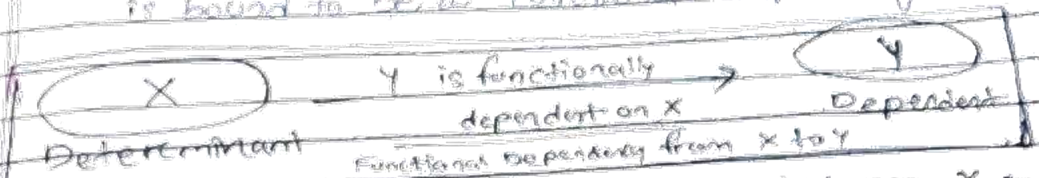
* Functional Dependency

→ In a relational database table, if an attribute is dependent on another attribute, it is called Functional Dependency.

→ Functional Dependency The set of constraints between two attributes in a table is called.

→ Form the value of one column, the value of another column can be determined ~~and~~ it is called Functional Dependency.

- If a column is the primary key then ~~o~~ there is bound to be a Functional Dependency.


X
Determinant
Y is functionally dependent on X
Functional Dependency from X to Y
Y
Dependent

Functional dependency between X and y

EX:

~~f~~ $d\ \alpha \rightarrow \beta$

$f(x) \rightarrow y$

$1 \rightarrow a$

$1 \rightarrow b$ ✗

→ $2 \rightarrow a$

	α	β
$t_1 \rightarrow$	a	1
$t_2 \rightarrow$	a	2
	c	3
	d	4

- If there is a functional dependency from alpha to beta then we can say that from alpha we can search the value of beta

EX:

$\alpha \subseteq R,\ \beta \subseteq R$

$\alpha \rightarrow \beta$

condition

- If $t_1[\alpha] = t_2[\alpha]$

$\hookrightarrow t_1[\beta] = t_2[\beta]$

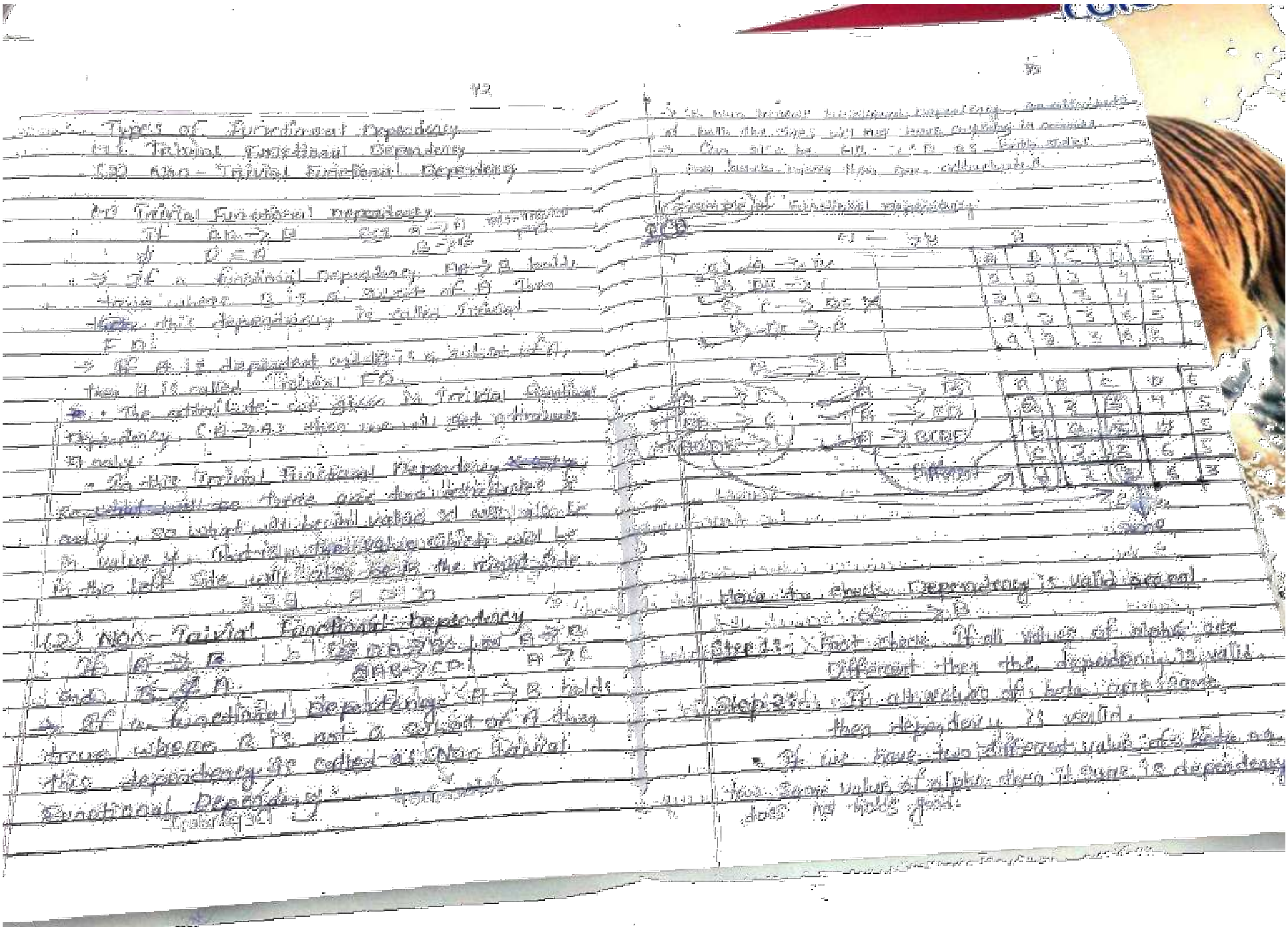
R

	α	β
$t_1 \rightarrow$	a	b
$t_2 \rightarrow$	a	b

$\alpha \longrightarrow \beta$

α ↓ determinent

β ↓ Dependent



42

Types of Functional Dependency
(1) Trivial Functional Dependency
(2) Non-Trivial Functional Dependency

(1) Trivial Functional dependency
If AB → B   Eg A → A
if B ⊆ A   B → B

⇒ If a functional Dependency AB → B holds true where B is a subset of A then this dependency is called Trivial F.D.

⇒ If A is dependent and B is a subset of A, then it is called Trivial FD.

• The attribute are given in Trivial functional dependency (A → A) then we will get attribute A only.

• In the Trivial Functional Dependency there are two attributes [illegible] so which attribute value of attribute [illegible] only, [illegible] value [illegible] in the right side in the left side [illegible]

A → B, A → A

(2) Non-Trivial Functional Dependency
If A → B   Eg AB → BC   A → B
and B ⊄ A   AB → CD   A → C

⇒ If a functional Dependency A → B holds true where B is not a subset of A then this dependency is called as Non Trivial Functional Dependency.

⇒ In non trivial functional dependency, no attribute of both the sides will not have anything in common.
⇒ Can also be AB → B as both sides [illegible] have more than one attribute.

Example of functional dependency

(Q) A → BC
BC → A
C → BC
[illegible] → B

A	B	C	D	E
[illegible]	3	2	4	[illegible]
3	4	3	4	5
8	3	3	6	5
9	3	3	6	[illegible]

A → B

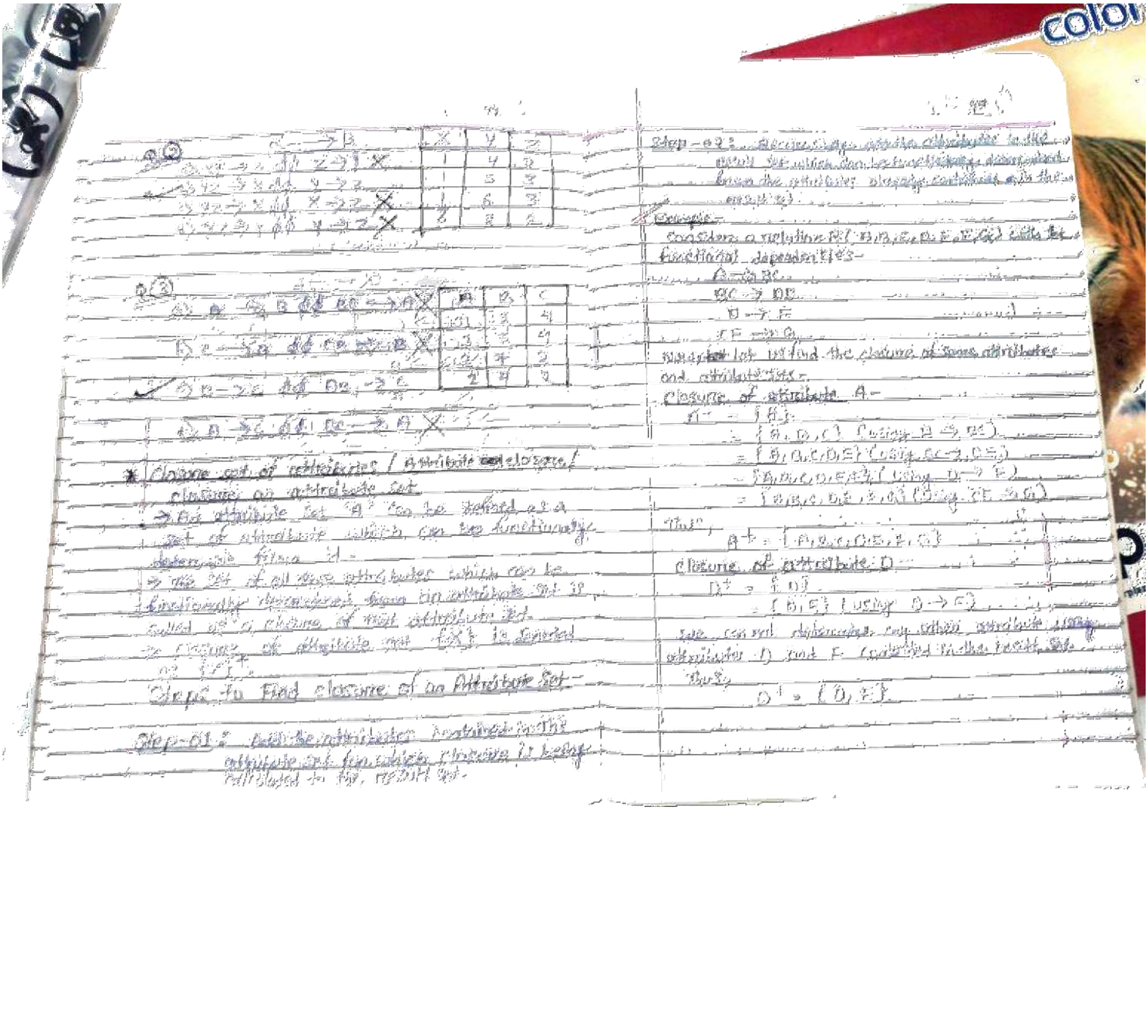
A → B, B → CD, A → BCDE

A	B	C	D	E
a	2	B	4	5
b	2	[illegible]	4	5
c	2	12	6	5
d	1	B	6	3

different

How to check Dependency is valid or not
α → β
Step 1:- First check if all values of alpha are different then the dependency is valid.
Step 2:- If all values of beta are same then dependency is valid.

• If we have two different value of beta on the same value of alpha then it [illegible] dependency does not holds good.



Q① X → Y

i) XY → Z and Z → Y ✗
ii) YZ → X and Y → Z ✓
iii) YZ → X and X → Z ✗
iv) XZ → Y and Y → Z ✗

X	Y	Z
1	4	2
1	5	3
1	6	3
2	2	2

Q② A → B

i) A → B and BC → A ✗
ii) C → B and CA → B ✗
iii) B → C and AB → C ✓
iv) A → C and BC → A ✗

A	B	C
1	1	4
1	2	4
3	3	2
3	3	2

* Closure set of attributes / Attribute closure / closure on attribute set

→ An attribute set "A" can be defined as a set of attribute which can be functionally determine from it.

→ The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set.

→ Closure of attribute set {X} is denoted as {X}⁺.

Steps to Find closure of an Attribute Set –

Step-01: Add the attributes contained in the attribute set for which closure is being calculated to the result set.

Step-02: Recursively add the attributes to the result set which can be functionally determined from the attributes already contained in the result set.

Example –

Consider a relation R(A, B, C, D, E, F, G) with the functional dependencies –

A → BC
BC → DE
D → F
CF → G

Now, let us find the closure of some attributes and attribute sets –

Closure of attribute A –

$A^+ = \{A\}$
$= \{A, B, C\}$ (using A → BC)
$= \{A, B, C, D, E\}$ (using BC → DE)
$= \{A, B, C, D, E, F\}$ (using D → F)
$= \{A, B, C, D, E, F, G\}$ (using CF → G)

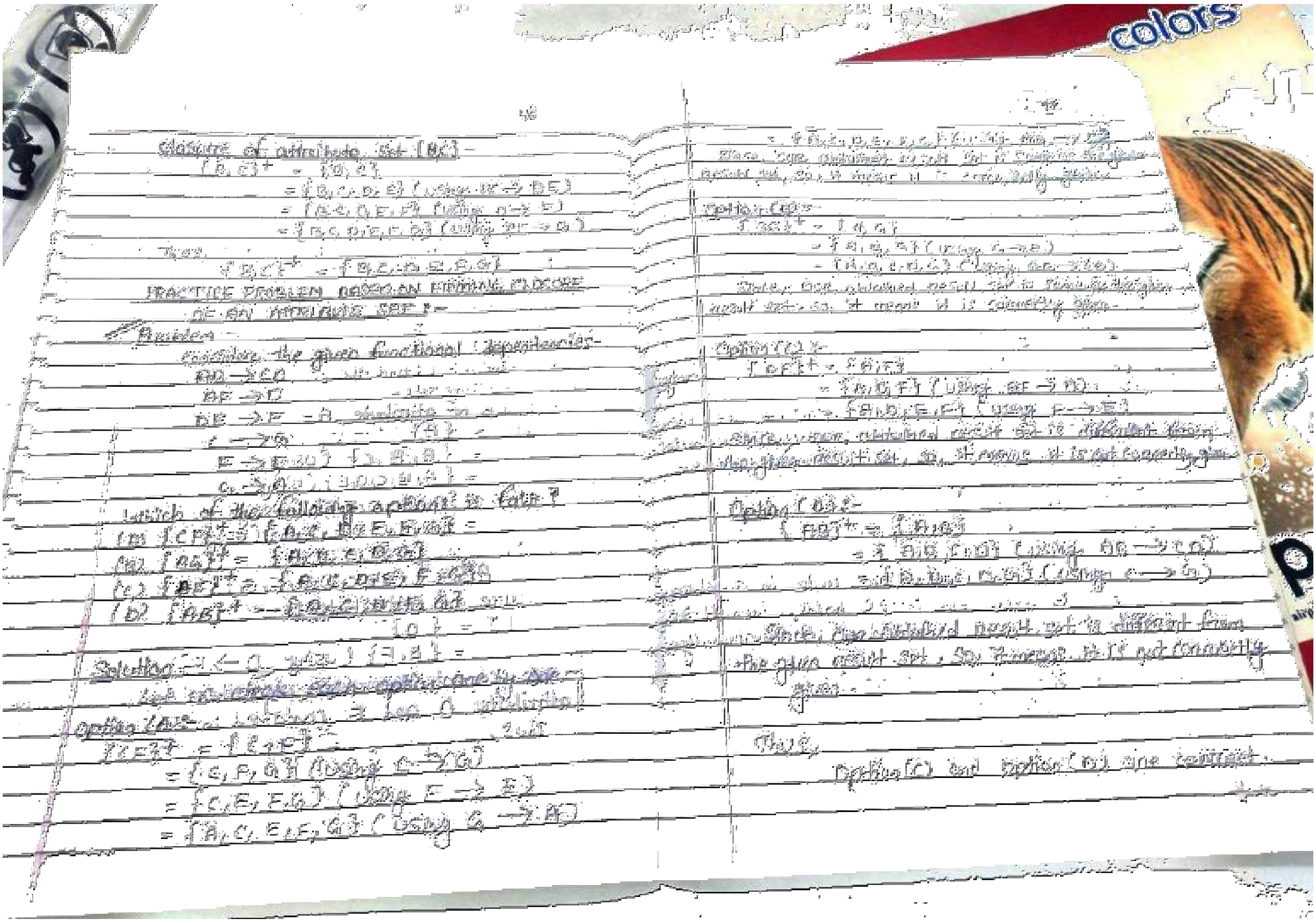
Thus,
$A^+ = \{A, B, C, D, E, F, G\}$

Closure of attribute D –

$D^+ = \{D\}$
$= \{D, F\}$ (using D → F)

We can not determine any other attribute using attributes D and F contained in the result set.

Thus,
$D^+ = \{D, F\}$



Closure of attribute set {BC}-

{B,C}+ = {B,C}

= {B,C,D,E} (Using BC → DE)

= {B,C,D,E,F} (Using D → F)

= {B,C,D,E,F,G} (Using BE → G)

Thus,

{B,C}+ = {B,C,D,E,F,G}

PRACTICE PROBLEM BASED ON FINDING CLOSURE OF AN ATTRIBUTE SET:-

Problem-

Consider the given functional dependencies-

AB → CD

AF → D

DE → F

C → G

F → E

G → A

Which of the following options is false?

(A) {CF}+ = {A,C,D,E,F,G}

(B) {BG}+ = {A,B,C,D,G}

(C) {AF}+ = {A,C,D,E,F,G}

(D) {AB}+ = {A,C,D,F,G}

Solution-

Let us check each option one by one-

Option (A)-

{CF}+ = {C,F}

= {C,F,G} (Using C → G)

= {C,E,F,G} (Using F → E)

= {A,C,E,F,G} (Using G → A)

= {A,C,D,E,F,G} (Using AF → D)

Since, our obtained result set is same as the given result set, so, it means it is correctly given.

Option (B)-

{BG}+ = {B,G}

= {A,B,G} (Using G → A)

= {A,B,C,D,G} (Using AB → CD)

Since, our obtained result set is same as the given result set, so, it means it is correctly given.

Option (C)-

{AF}+ = {A,F}

= {A,D,F} (Using AF → D)

= {A,D,E,F} (Using F → E)

Since, our obtained result set is different from the given result set, so, it means it is not correctly given.

Option (D)-

{AB}+ = {A,B}

= {A,B,C,D} (Using AB → CD)

= {A,B,C,D,G} (Using C → G)

Since, our obtained result set is different from the given result set, so, it means it is not correctly given.

Thus,

Option (C) and Option (D) are incorrect.

* Armstrong Axiom / rule :-

→ Axiom : It is a statement that is taken to be true and serve as a premise or starting point for further argument.

→ Armstrong axioms hold on every relational database and can be used to generate closure set.

Primary Rules

(1) Reflexivity :-

If A is a set of attributes and B is subset of A, then A holds B.

(If $B \subseteq A$ then $A \rightarrow B$) (This property is trivial property.)

(2) Augmentation :-

If $A \rightarrow B$ holds and Y is attribute set, then $AY \rightarrow BY$ also holds. That is adding attributes in dependencies, does not change the basic dependencies.

(If $A \rightarrow B$ then $AC \rightarrow BC$)

(3) Transitivity :-

Same as the transitive rule in algebra, if $A \rightarrow B$ holds and $B \rightarrow C$ holds, then $A \rightarrow C$ also holds. $A \rightarrow B$ is called as A functionally determines B.

(If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$)

Secondary Rules

(1) Union :-

If $A \rightarrow B$ holds and $A \rightarrow C$ holds, then $A \rightarrow BC$ holds.

(If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$)

(2) Composition :-

(If $X \rightarrow Y$ and $Z \rightarrow W$ then $XZ \rightarrow YW$) ($XZ \rightarrow YZ$, $XZ \rightarrow YW$, $XZ \rightarrow ZY$ [illegible])

(3) DeComposition :-

If $A \rightarrow BC$ holds then $A \rightarrow B$ and $A \rightarrow C$ hold.

(If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$)

(4) Pseudo Transitivity :-

If $A \rightarrow B$ holds and $BC \rightarrow D$ holds, then $AC \rightarrow D$ holds.

(If $X \rightarrow Y$ and $WY \rightarrow Z$ Then $WX \rightarrow Z$) ($WX \rightarrow$ $WY \rightarrow$ $WY \rightarrow$

* Equivalence of Functional Dependencies

→ Two different sets of functional dependen given relation may or may not be e

→ If F and G are the two sets of fu dependencies, then following 3 cases

Case-01 :- F covers G (F ⊇ G)
Case-02 :- G covers E (G ⊇ F)
Case-03 :- Both F and G cover each other (F = G)

• Case-01 :- Determining whether F covers G-

Following steps are followed to determine whether F covers G or not-

Step-01 :

→ Take the functional dependencies of set G into consideration.

→ For each functional dependency X→Y find the closure of X using the functional dependencies of set G.

Step-02 :

→ Take the functional dependencies of set G into consideration.

→ For each functional dependency X→Y find the closure of X using the functional dependencies of set F.

Step-03 :

Compare the results of step-01 and step-02.
If the functional dependencies of set F has determined all those attributes that were determined by the functional dependencies of set G, then it means F covers G.
Thus, we conclude F covers G (F ⊇ G)
otherwise not.

• Case-02 :- Determining whether G covers F -

Following steps are followed to determine whether G covers F or not-

Step-01 :

→ Take the functional dependencies of set F into consideration.

→ For each functional dependency X→Y, find the closure of X using the functional dependencies of set F.

Step-02 :

→ Take the functional dependencies of set F consideration.

→ For each functional dependency X→Y, find closure of X using the functional dependen of set G.

Step-03 :

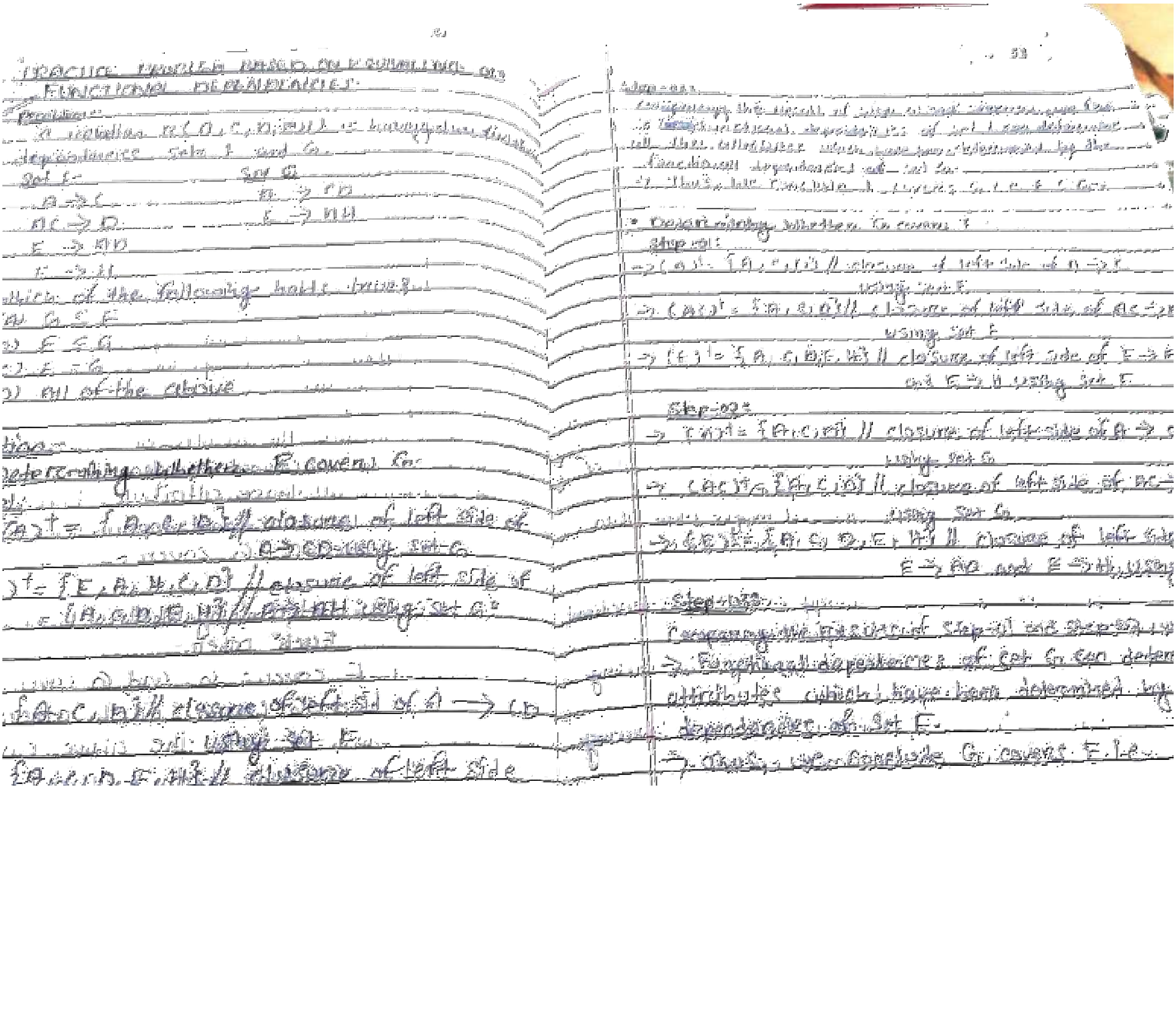
→ Compare the results of step-01 and step

→ If the functional dependencies of set G has mined all those attributes that were d using the functional dependencies of se then it means G covers F.

• Case-03 : Determining whether Both F and Each other-

→ If F covers G and G covers F, F and G cover each other.

→ Thus, if both the above cases we conclude both F and G cov



PRACTICE PROBLEM BASED ON EQUIVALENCE OF FUNCTIONAL DEPENDENCIES:

Problem:-

A relation R(A, C, D, E, H) is having two functional dependencies sets F and G.

Set F:-	Set G
A → C	A → CD
AC → D	E → AH
E → AD	
E → H	

Which of the following holds true?

(a) G ⊆ F

(b) F ⊆ G

(c) F = G

(d) All of the above

Solution:-

Determining whether F covers G

Step-01: [illegible]

$(A)^+ = \{A, C, D\}$ // closure of left side of A → CD using set G

$(E)^+ = \{E, A, H, C, D\}$ // closure of left side of E → AH using set G

$= \{A, C, D, E, H\}$ // E → AH using set G

Step-02: [illegible]

$(A)^+ = \{A, C, D\}$ // closure of left side of A → CD using set F

$(E)^+ = \{A, C, D, E, H\}$ // closure of left side

53

Step-03:

Comparing the results of step-01 and step-02, we find functional dependencies of set F can determine all the attributes which have been determined by the functional dependencies of set G.

∴ Thus, we conclude F covers G i.e. F ⊇ G.

* Determining whether G covers F

Step-01:

⇒ $(A)^+ = \{A, C, D\}$ // closure of left side of A → C using set F

⇒ $(AC)^+ = \{A, C, D\}$ // closure of left side of AC → D using set F

⇒ $(E)^+ = \{A, C, D, E, H\}$ // closure of left side of E → AD and E → H using set F

Step-02:

⇒ $(A)^+ = \{A, C, D\}$ // closure of left side of A → C using set G

⇒ $(AC)^+ = \{A, C, D\}$ // closure of left side of AC → D using set G

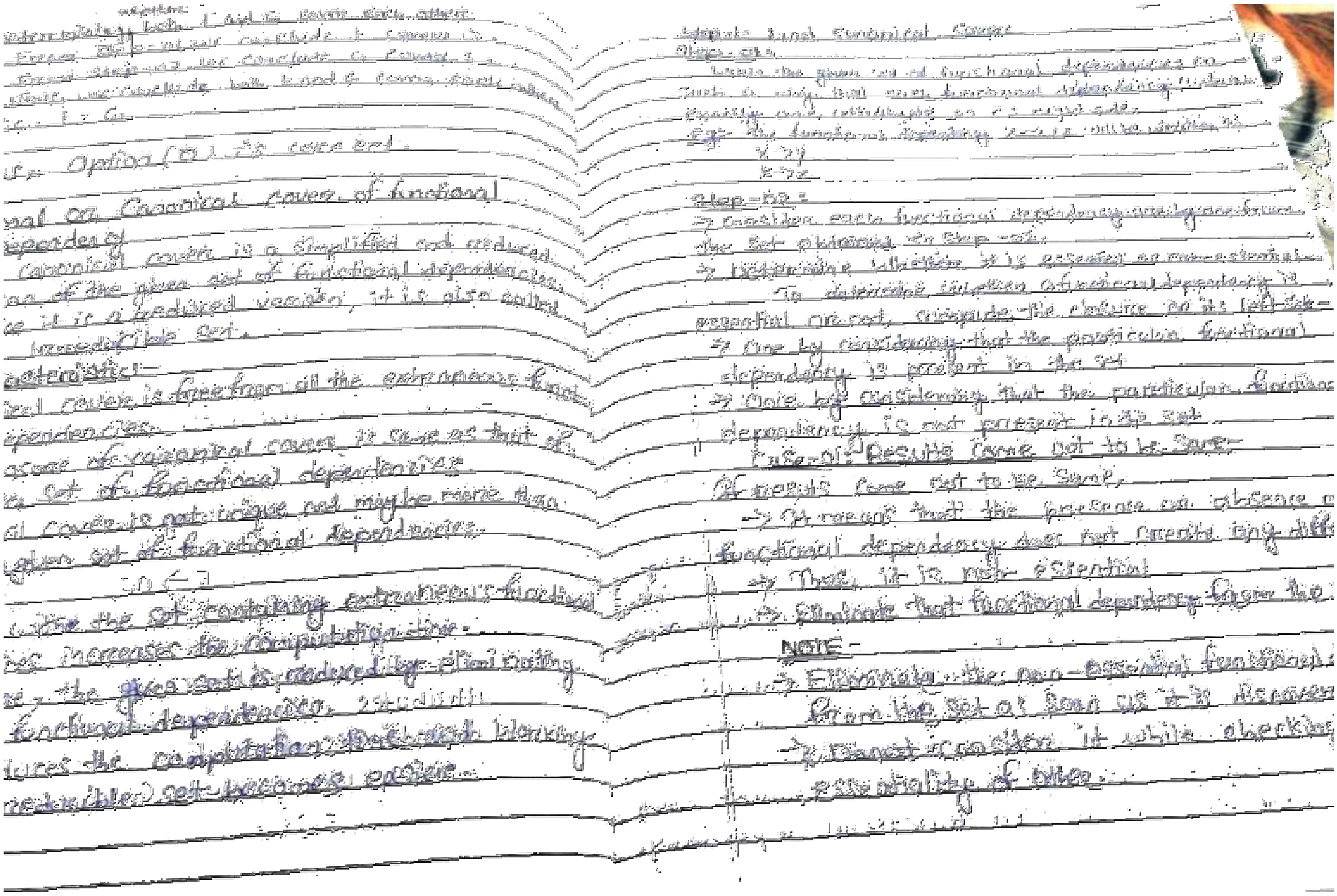
⇒ $(E)^+ = \{A, C, D, E, H\}$ // closure of left side of E → AD and E → H using

Step-03:

Comparing the results of step-01 and step-02, we find

⇒ Functional dependencies of set G can determine all attributes which have been determined by dependencies of set F.

⇒ Thus, we conclude G covers F i.e.



whether
determining both I and G cover each other
From (B)-cover we conclude I covers G
From step-02 we conclude G covers I
Thus, we conclude both I and G cover each other
i.e. I ≡ G

∴ Option (D) is correct.

…nal or Canonical cover of functional
dependency

Canonical cover is a simplified and reduced
…on of the given set of functional dependencies.
…ce it is a reduced version, it is also called
Irreducible set.

…acteristic:-

…cal cover is free from all the extraneous funct…
…ependencies.
…osure of canonical cover is same as that of
…osure of given set of functional dependencies.
…e set of functional dependencies
…al cover is not unique and may be more than
… given set of functional dependencies.

Need -

…ing the set containing extraneous functional
…es increases the computation time.
…e the given set is reduced by eliminating
… functional dependencies, [illegible]
… functional dependencies [illegible]
…duces the computation [illegible]
…reducible set becomes easier.

Steps to find Canonical Cover

Step-01:

Write the given set of functional dependencies in
such a way that each functional dependency contains
exactly one attribute on its right side.
Eg: The functional dependency X→YZ will be written as
X→Y
X→Z

Step-02:

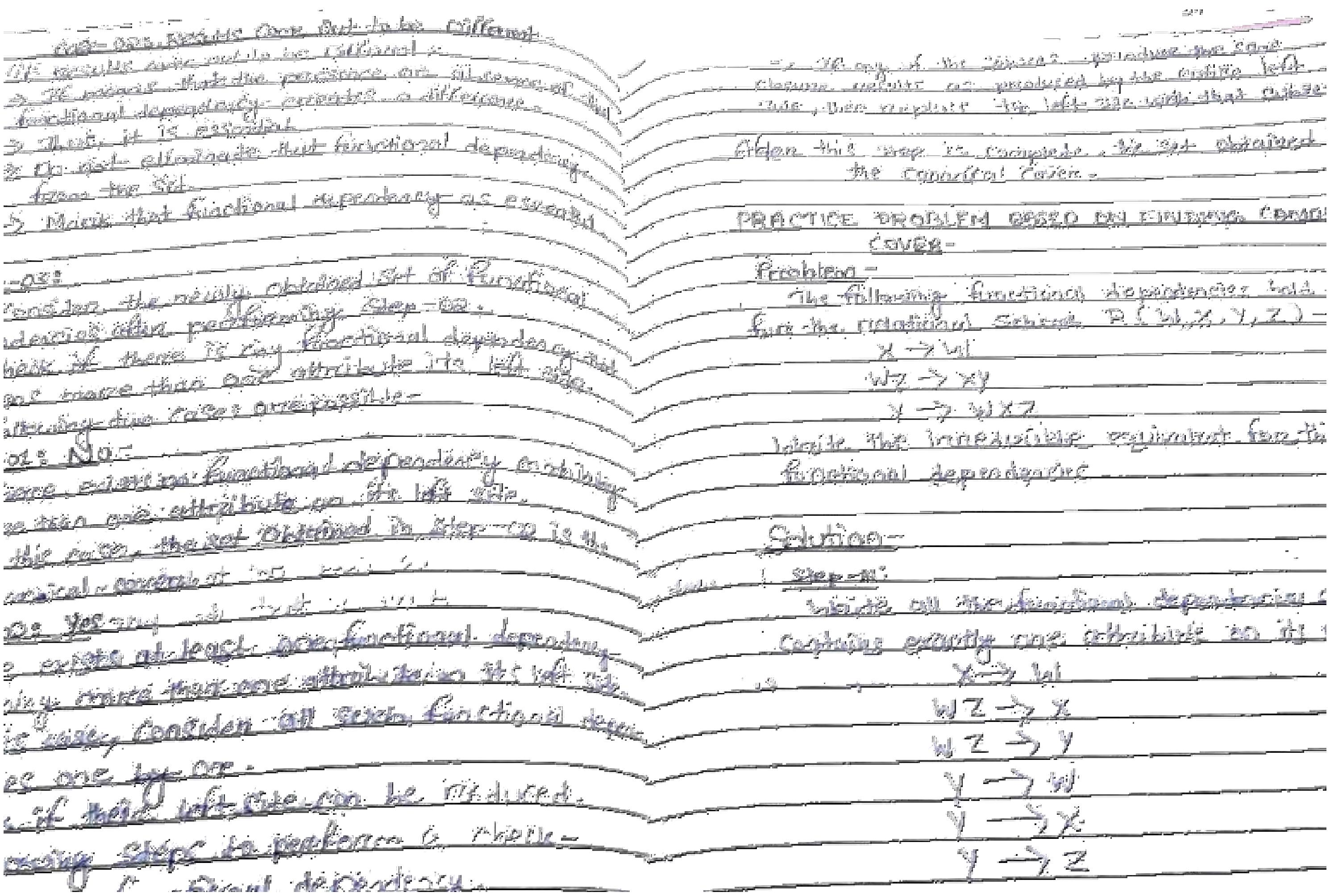
⇒ Consider each functional dependency one by one from
the set obtained in Step-01.
⇒ Determine whether it is essential or non-essential.
To determine whether a functional dependency is
essential or not, compute the closure of its left side-
⇒ Once by considering that the particular functional
dependency is present in the set
⇒ Once by considering that the particular functional
dependency is not present in the set

Case-01: Results come out to be Same-

If results come out to be same,
→ It means that the presence or absence o…
functional dependency does not create any diff…
⇒ Thus, it is not essential
⇒ Eliminate that functional dependency from the …

NOTE-

⇒ Eliminate the non-essential functional …
from the set as soon as it is discover…
⇒ Donot consider it while checking
essentiality of other.



Case-02: Results come out to be different

If results come out to be different -

=> It means that the presence or absence of that functional dependency creates a difference.

=> Thus, it is essential

=> Do not eliminate that functional dependency from the set.

=> Mark that functional dependency as essential

-03:

onsider the newly obtained set of functional dencies after performing Step-02.

heck if there is any functional dependency that ng more than one attribute its left side.

llowing two cases are possible-

01: No-

ere existing functional dependency containing re than one attribute on its left side.

this case, the set obtained in Step-02 is the onical cover.

02: Yes

e exists at least one functional dependency ning more than one attribute on its left side.

is case, consider all such functional depen es one by one.

if their left side can be reduced.

owing steps to perform a check-

functional dependency.

=> If any of the subsets produce the same closure results as produced by the entire left side, then replace the left side with that subse

After this step is complete, the set obtained the canonical cover.

PRACTICE PROBLEM BASED ON FINDING CANO COVER-

Problem-

The following functional dependencies hold for the relational schema R (W, X, Y, Z) -

$X \rightarrow W$

$WZ \rightarrow XY$

$Y \rightarrow WXZ$

Write the irreducible equivalent for th functional dependencies

Solution-

Step-01:

Write all the functional dependencies c contains exactly one attribute on its

$X \rightarrow W$

$WZ \rightarrow X$

$WZ \rightarrow Y$

$Y \rightarrow W$

$Y \rightarrow X$

$Y \rightarrow Z$

Step-02:

Check the essentiality of each functional dependency one by one.

For $X \to W$:

- Considering $X \to W$

$(X)^+ = \{X, W\}$

- Ignoring $X \to W$

$(X)^+ = \{X\}$

Now,

⇒ Clearly, the two results are different.

⇒ Thus, we conclude that $X \to W$ is essential and can not be eliminated.

For $WZ \to X$:

- Considering $WZ \to X$

$(WZ)^+ = \{W, X, Y, Z\}$

- Ignoring $WZ \to X$

$(WZ)^+ = \{W, X, Y, Z\}$

Now,

⇒ Two results are same.

⇒ Thus, we conclude that $WZ \to X$ is non-essential and can be eliminated.

[illegible]

[illegible] $WZ \to X$, functional dependencies reduces to -

$X \to W$

$WZ \to Y$

$Y \to W$

$Y \to X$

For $WZ \to Y$:

- Considering $WZ \to Y$

$(WZ)^+ = \{W, X, Y, Z\}$

- Ignoring $WZ \to Y$

$(WZ)^+ = \{W, Z\}$

Now,

⇒ Two results are different.

⇒ Thus, we conclude that $WZ \to Y$ is essential and can not be eliminated.

For $Y \to W$:

- Considering $Y \to W$

$(Y)^+ = \{W, X, Y, Z\}$

- Ignoring $Y \to W$

$(Y)^+ = \{W, X, Y, Z\}$

Now,

⇒ Two results are same.

⇒ Thus we conclude that $Y \to W$ is non-essential and can be eliminated.

Eliminating $Y \to W$, functional dependencies reduces to -

$X \to W$

$WZ \to Y$

$Y \to X$

$Y \to Z$

For $Y \to X$:

- Considering $Y \to X$

$(Y)^+ = \{W, X, Y, Z\}$

- Ignoring $Y \to X$

Now,

⇒ Two results are different

⇒ Thus, we conclude that $Y \to X$ is essential and can not be eliminated.

For, $Y \to Z$:

- Considering $Y \to Z$

$(Y)^+ = \{W, X, Y, Z\}$

- Ignoring $Y \to Z$

$(Y)^+ = \{W, X, Y\}$

⇒ Two results are different.

⇒ Thus, we conclude that $Y \to Z$ is essential and can not be eliminated.

[illegible] here, our essential functional dependencies are:

$X \to W$

$WZ \to Y$

$Y \to X$

$Y \to Z$

[illegible]-03

[illegible] the functional dependencies having more than one attribute on their left side.

[illegible] their left side can be reduced.

[illegible] set,

$WZ \to Y$ contains more than one attribute on left side.

Considering $WZ \to Y$

$(WZ)^+ = \{W, X, Y, Z\}$

⇒ Check if the closure result of any subset matches to the closure result of WZ.

$(W)^+ = \{W\}$

$(Z)^+ = \{Z\}$

Clearly,

⇒ None of the subsets have the same closure result same as that of the entire left side.

⇒ Thus, we conclude that we can not write WZ as $W \to Y$ or $Z \to Y$.

⇒ Thus, set of functional dependencies obtained in step-03 is the canonical cover.

Finally, the canonical cover is –

$X \to W$

$WZ \to Y$

$Y \to X$

$Y \to Z$

* Types of Keys

(1) Super key :-

A super key is a combination of all p attributes that can uniquely identify the r tuple) in the given relation.

⇒ Super key is a super set of a can

⇒ A table can have many super ke

⇒ A super key ~~key~~ may have add attribute that are not needed fo identify

All the attributes in a super key a essential to identify each tuple uniquely

Example

[illegible]

[illegible]

E → [illegible]

[illegible] super key [illegible]

[illegible] = ECDFABD

[illegible] = AB.

[illegible] super key [illegible]

Candidate key:-

[illegible] super key is called as a candidate key.

[illegible] a minimal attribute(s) that can identify [illegible] uniquely in the given relation.

[illegible] a candidate key [illegible]

The attributes in a candidate key are [illegible] necessary to identify each tuple uniquely.

[illegible] attribute from the candidate key [illegible] identifying each tuple uniquely.

[illegible]

[illegible]

[illegible]

Finding Candidate keys-

[illegible]

Step-01:

→ Determine all essential attributes of the given relation.

→ Essential attributes are those attributes which are not present on RHS of any functional dependency.

→ Essential attributes are always a part of candidate key

→ This is because they can not be determined by other attributes.

Example

Let R(A,B,C,D,E,F) be a relation scheme with the following functional dependencies-

A → B

C → D

D → E

Here, the attributes which are not present on RHS of any functional dependency are A, C [illegible]

So, essential attributes are - A, C [illegible]

Step-03:

→ The remaining attributes of the relation are non-essential attributes.

→ This is because they can be determined by using essential attributes.

And following two cases

Case-01:

If all essential attributes together can determine all remaining non-essential attributes, then

→ the combination of essential attributes is the candidate key

→ It is the only possible candidate key.

Case-02:

If all essential attributes together can not determine all remaining non-essential attributes, then,

→ The set of essential attributes and some non-essential attributes will be the candidate keys.

In this case, multiple candidate keys are possible.

To find the candidate keys, we check different combinations of essential and non-essential attributes.

[illegible]

ple-1

R(A, B, C, D, E, F) be a relation scheme with following dependencies—

C → F

E → A

E C → D

A → B

Which of the following is a key for R.

(1) CD

(2) EC

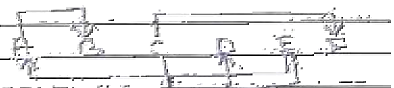
(3) AE

(4) AC

Also, determine the total number of candidate keys.

Solution:

R =



Essential attributes of the relation are C and E.

Find the closure of CE.

So, we have

{CE}+

= {C, E, F, A, D, B}

We conclude that CE can determine all attributes of the given relation.

So, CE is the only possible candidate key of the relation.

Thus, option (2) is correct.

Total number of candidate keys—

Only one candidate key CE.

R = (A , B , C , D) find the total number of candidate key. A → B , B → C , C → A

Solution

First determine all essential attribute of the given relation.

R = (A B C D)

So, essential attributes are = D.

closure of D

$(D)^+ = D$

then, combination of non-essential attribute

$(AD)^+ = ABCD$

$(BD)^+ = BDCA$

$(CD)^+ = CDAB$

- So, AD, BD and CD only the possible candidate key of the relation.

→ Because already all the minimal structure which were the size of two or two have all become candidate keys. Now if we add any of it will be the superkey. So it can be a super key but not a candidate key because a candidate key must be minimal.

Example-03

R = (A, B, C, D) find total number of candidate key using closure

AB → CD

D → A

Solution:

R (A B C D)

So, essential attribute are = B

Closure of B

$(B)^+ = B$

then, combination of non essential attribute

$(AB)^+ = BACD$

$(BC)^+ = BC$

$(BD)^+ = BDAC$

So AB, BD only possible candidate of the relation.

Example-04

R = (A B C D E F) find the total of candidate key.

AB → C

C → D

B → AE

Solution:

R (A B C D E F)

So, essential attribute are = B F

closure of BF

$(BF)^+ = BFAECD$

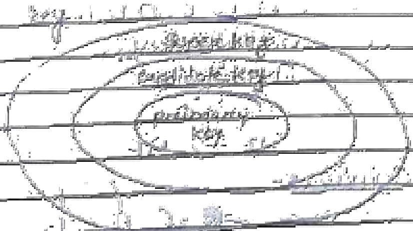
We conclude that BF can determine attribute of the given relation.

So, BF is the only possible c

the relation.

(iv) Primary key :-
A primary key is a candidate key that the database designer select while designing the database.
OR
Candidate key that the database designer implements is called as a primary key.

NOTES:-
- The value of primary key can never be NULL.
- The value of primary key must always be unique.
- The values of primary key can never be changed i.e. no updation is possible.
- The value of primary key must be assigned when inserting a record.
- A relation is allowed to have only one primary key.



IMPORTANCE OF NORMALIZATION

Student information

Name	Age	B.code	B.Name	HOD name
A	18	101	CS	XYZ
B	19	101	CS	XYZ
C	18	101	CS	XYZ
D	21	102	EC	PQR
[illegible]	[illegible]	102	[illegible]	PQR

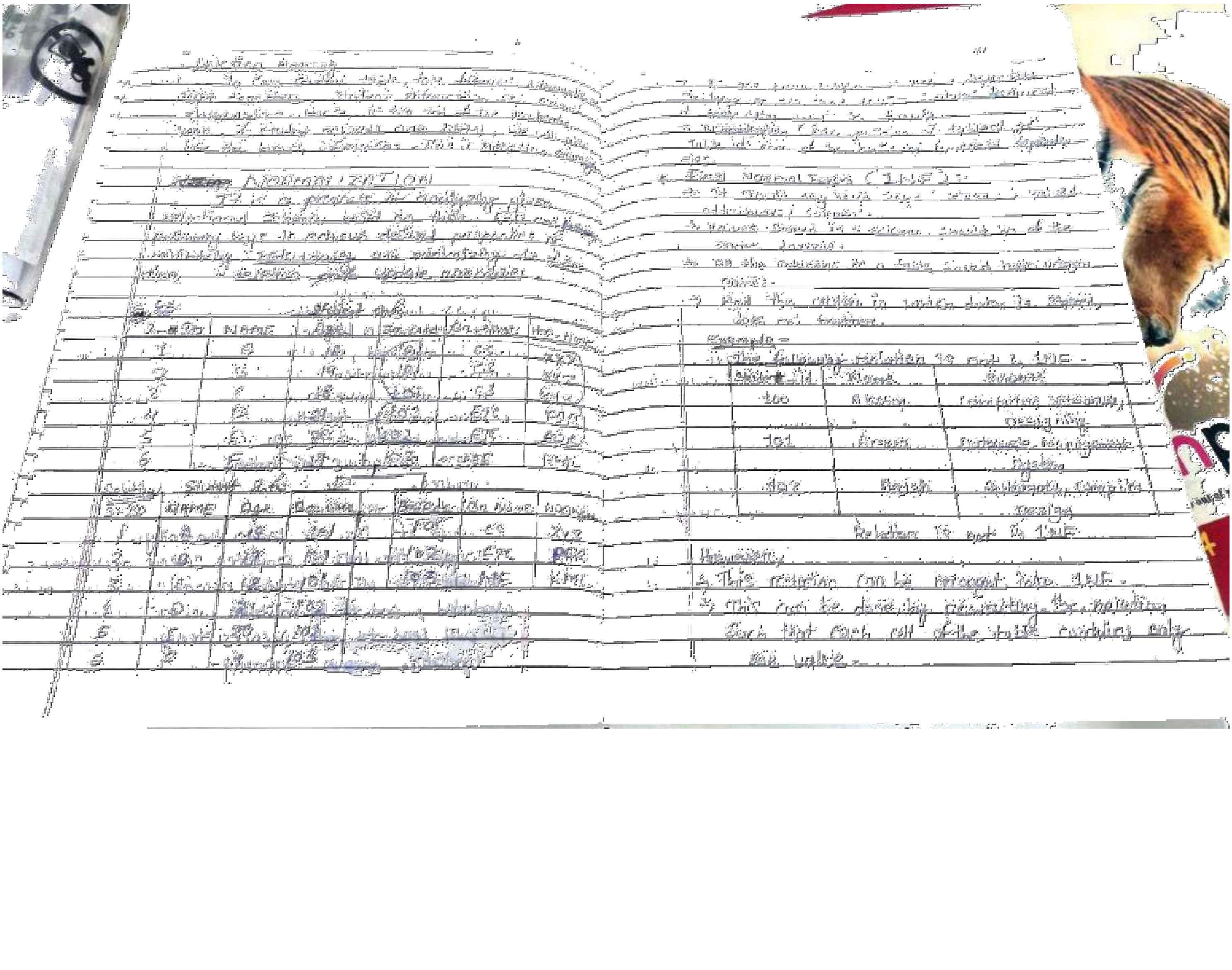
Redundancy

Here, in the table student information we have tried to store entire data about student. Result → entire branch data of a branch must be repeated for every student of the branch.
Redundancy → is when same data is stored multiple times unnecessarily in a database.

Disadvantages →
(i) Insertion, Updation and deletion anomalies.
(ii) Inconsistency (data)
(iii) Increase in database size and increase in time (slow).

Problems without normalization
(i) Insertion Anomaly
Suppose, for a new admission, until and unless student opts for a branch, data of the student cannot be inserted, or else we will have to set branch information as NULL.
Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.
These scenarios are nothing but Insertion anomalies.

(ii) Updation Anomaly
What if Mr. XYZ leaves the college? or no longer the HOD of computer science department? In that case all the student records will be updated, and if by mistake we miss any it will lead to data inconsistency.



Deletion Anomaly

→ [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

NORMALIZATION

It is a process of analyzing given relational schema based on their FDs and primary keys to achieve desired properties of minimizing redundancy and minimizing the insertion, deletion and update anomalies.

S_NO	NAME	Age	Branch code	Branch	Hostel name
1	A	18	101	CS	XYZ
2	B	19	101	CS	XYZ
3	C	18	101	CS	XYZ
4	D	21	102	EC	PQR
5	E	20	102	EC	PQR
6	F	20	103	ME	KML

S_NO	NAME	Age	[illegible]	Branch code	Branch name	Hostel
1	A	18	[illegible]	101	CS	XYZ
2	B	19	[illegible]	102	EC	PQR
3	C	20	[illegible]	103	ME	KML
4	D	21	[illegible]	[illegible]	[illegible]	
5	E	20	[illegible]	[illegible]	[illegible]	
6	F	19	[illegible]	[illegible]	[illegible]	

→ It are some steps [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

→ Normalization [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

First Normal Form (1NF):

→ It should only have single (atomic) valued attributes/columns.

→ Values stored in a column should be of the same domain.

→ All the columns in a table should have unique names.

→ And the order in which data is stored does not matter.

Example –

i) The following relation is not in 1NF –

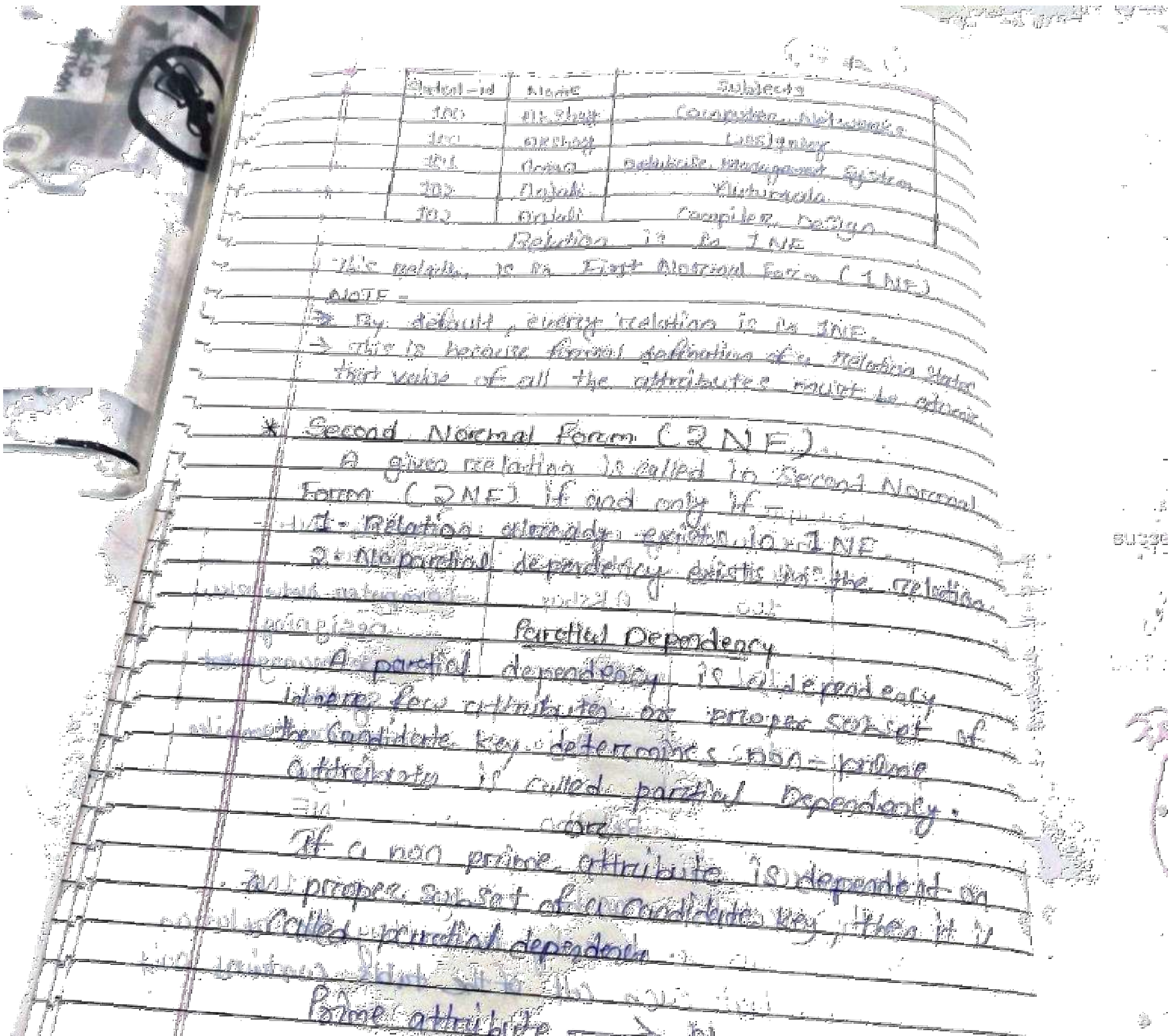
Student_id	Name	Subjects
100	Akash	Computer Networks, Designing
101	Aravind	Database Management System
102	Rajesh	Automata, Compiler Design

Relation is not in 1NF

However,

→ This relation can be brought into 1NF.

→ This can be done by rewriting the relation such that each cell of the table contains only one value.



Student-id	Name	Subjects
100	Akshay	Computer Networks
100	Akshay	Designing
101	Aman	Database Management System
102	Anjali	Multimedia
102	Anjali	Compiler Design

Relation is in 1NF

This relation is in First Normal Form (1NF)

NOTE -

→ By default, every relation is in 1NF.

→ This is because formal definition of a relation states that value of all the attributes must be atomic.

* Second Normal Form (2NF).

A given relation is called in Second Normal Form (2NF) if and only if -

1. Relation already exists in 1NF

2. No partial dependency exists in the relation.

Partial Dependency

A partial dependency is a dependency where few attributes or proper subset of the candidate key determines non-prime attributes is called partial dependency.

or

If a non prime attribute is dependent on a proper subset of a candidate key, then it is called partial dependency.

Prime attribute ⟶ Non prime attribute

P → NP

In other words,

A → B is called a partial dependency if and only if-

1. A is a subset of some candidate key
2. B is a non-prime attribute.

If any one condition fails, then it will not be a partial dependency.

Fully Functional Dependency

ABC → D { D is fully FD on ABC }

↑ FD

{ D cannot dependents on any subset of ABC }

These are the subset of ABC

{ BC → D
C → D
A → D

not possible because
BC cannot determine D
C cannot determine D
A cannot determine D

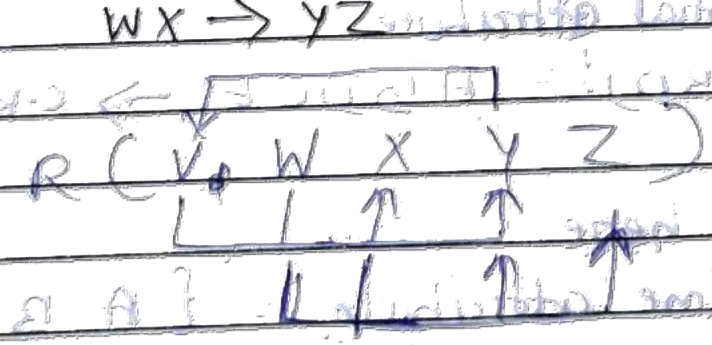
only ABC determine D
means D is fully F.D on ABC

Example-1 (2NF)

Consider a relation R (V, W, X, Y, Z) with functional dependencies

VW → XY
Y → V
WX → YZ

Solution



XW → W

Y → V essential attribute

WX → Y X (W)⁺ = W

$(WV)^+ = WVXYZ \rightarrow C.K$

$(WX)^+ = WXYZV \rightarrow C.K$

$(WY)^+ = WYVXZ \rightarrow C.K$

$X\ (WZ)^+ = WZ$

From here,

- Prime attributes = {V, W, X, Y}
- Non-prime attribute = {Z}

Now, if we observe the given dependencies -

- This is because there exists no dependency where incomplete candidate key determines any non prime attribute.
- There is no partial dependency.

∴ Thus, we conclude that the given relation is in 2NF.

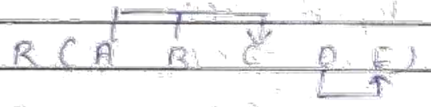
Example - 2

Consider a relation R (A B C D E) with functional dependencies.

AB → C

D → E

Solution



essential attribute

$(ABD)^+ = ABDCE \rightarrow C.K$

AB → C

D → E

From here

- Prime attribute = {A B D}
- non prime attribute = {C E}

So here is the partial dependency exist

(A,B) - AB → C

(A,D) - D → E

So, there will be three tables below:

- R₁ (A B C)
- R₂ (D E)
- R₃ (A B D)

Example - 3

Consider a relation R (A B C D E F G H I J)

with Functional dependencies

AB → C

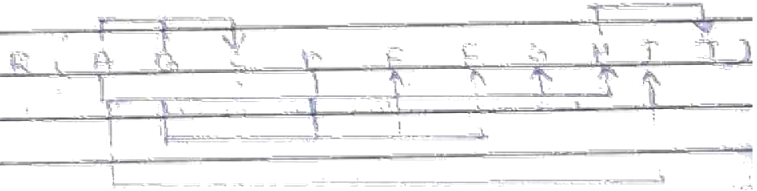
AD → GH

BD → EF

A → I

H → J

Solution



Essential attribute

$(ABD)^+ = ABDCGHEFIJ$

from here,

- Prime attribute = {A B D}
- Non prime attribute = {C G H E F I J}

⇒ Now is the partial dependency

(A B) — AB → C

(A B) — AD → [illegible]

(A D) — [illegible] → EF [illegible]

(A D) — A → I

— H → J

So there will be five tables here.

R1 (A B C [illegible])

R2 (A D [illegible])

R3 (B D E F) [illegible]

R4 (A I J [illegible])

R5 (A B D) [illegible]

* Third Normal Form (3NF) -

A given relation is called in Third Normal Form (3NF) if and only if—

1. Relation already exists in 2NF.
2. No transitive dependency exists for non-prime attributes.

OR

Any one condition holds for each non-trivial functional dependency A → B.

1. A is a Super Key
2. B is a prime attribute

Transitive Dependency

A → B is called a transitive dependency if and only if —

1. A is not a super key.
2. B is a non-prime attribute.

If any one condition fails, then it is not a transitive dependency.

(Non-prime attribute → Non-prime attribute
NP → NP)

NOTE -

⇒ Transitive dependency must not exist for non-prime attributes.

⇒ However, transitive dependency can exist for prime attributes.

Example-1 (3NF)

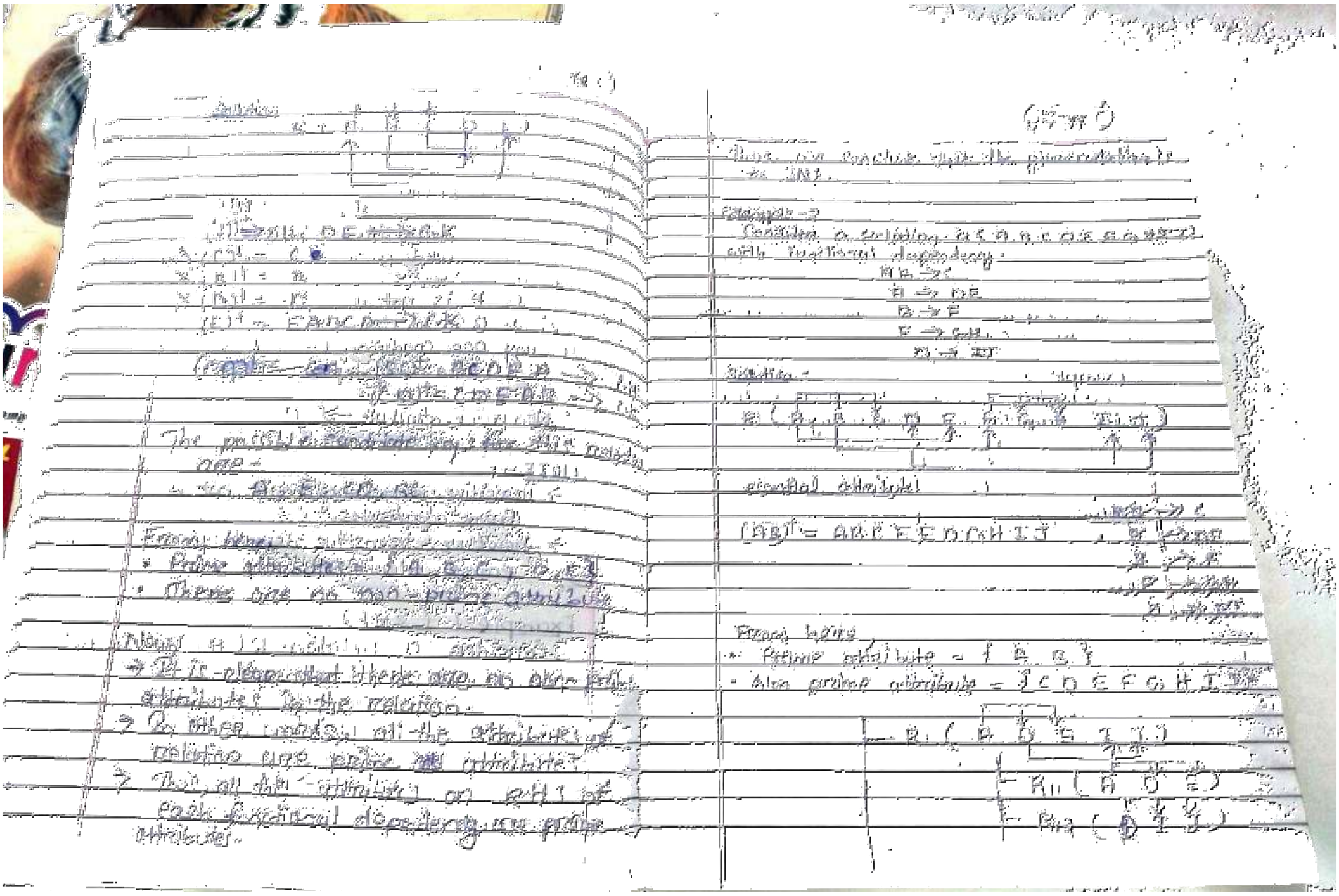
Consider a relation - R(A, B, C, D, E) with functional dependencies-

A → BC

CD → E

B → D

E → A



Solution:

R (A B C D E)

(A)⁺ = A B C D E [illegible]

(C)⁺ = C

(B)⁺ = B

(D)⁺ = D

(E)⁺ = E A B C D [illegible]

Candidate key [illegible] A, E, C D [illegible]

The possible candidate keys for this relation are -

A, E, CD, BC [illegible]

From here [illegible]

- Prime attributes = { A, B, C, D, E }
- There are no non-prime attributes

Now, [illegible]

→ It is clear that there are no non-prime attributes in the relation.

→ In other words, all the attributes of relation are prime attributes.

→ Thus, all the attributes on RHS of each functional dependency are prime attributes.

Thus, we conclude that the given relation is in 3NF.

Example →

Consider a relation R (A B C D E F G H I J) with functional dependency.

AB → C

A → DE

B → F

F → GH

D → IJ

Solution -

R (A B C D E F G H I J)

[illegible] attribute

(AB)⁺ = A B C F E D G H I J

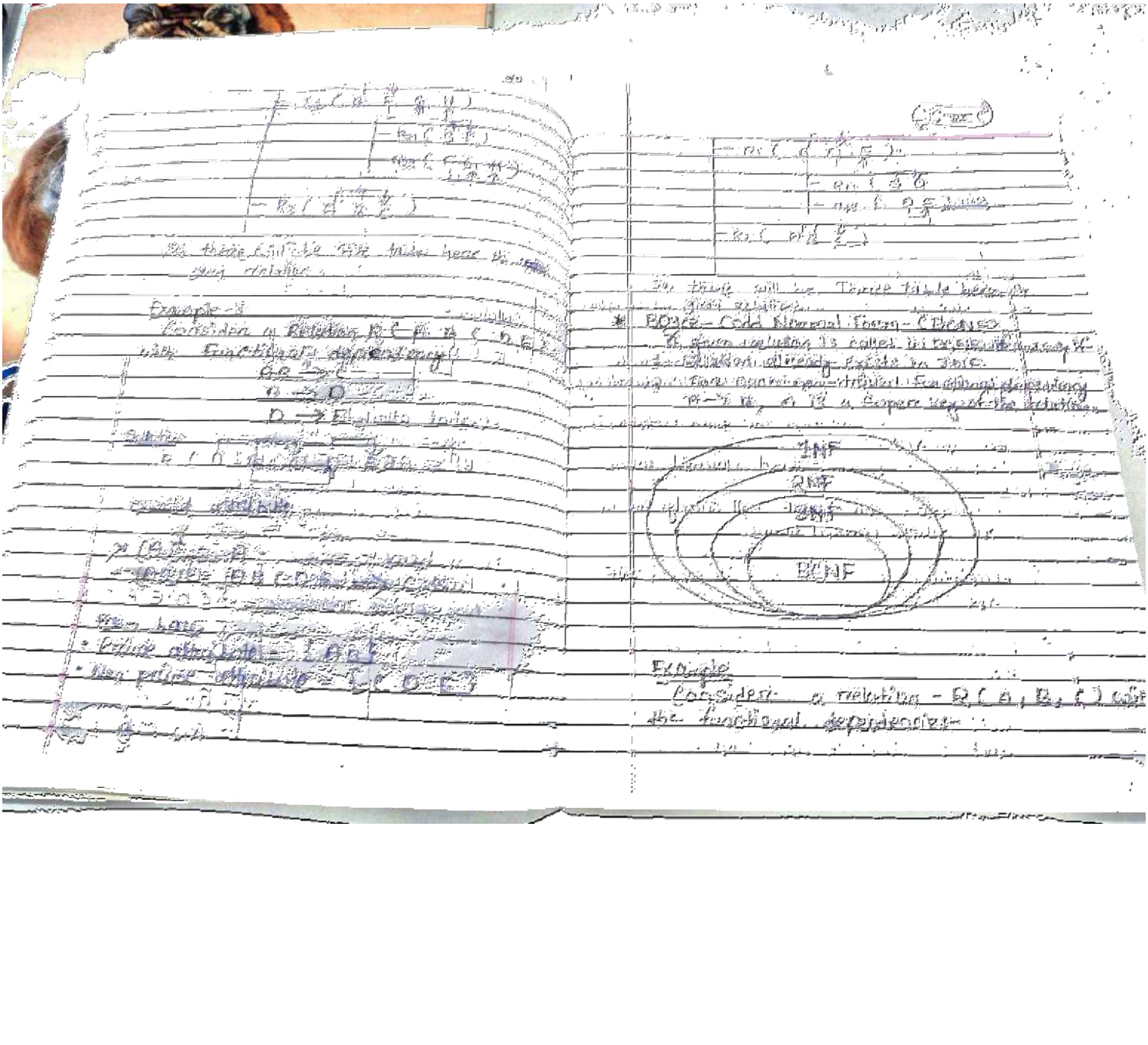
From here

- Prime attribute = { A B }
- Non prime attribute = { C D E F G H I J }

R₁ (A D E I J)

R₁₁ (A D E)

R₁₂ (D I J)



R_4 (A E G H)

R_{41} (G H)

R_{42} (E G H)

R_2 (A B E)

So there will be [illegible] table here [illegible] given relation.

Example - 3

Consider a Relation R (A B C D E) with Functional dependency :

AB → C

B → D

D → E [illegible]

Solution

R (A B C D E) [illegible]

[illegible] attribute

→ (AB)⁺ = ABCDE [illegible]

[illegible] (AB C D E) [illegible]

[illegible]

Min key [illegible]

• Prime attribute = { A B }

• Non prime attribute = { C D E }

R_1 (A B C)

R_{11} (B D)

R_{12} (D E) [illegible]

R_2 (A B)

So there will be Three table here for given relation.

* Boyce - Codd Normal Form - (BCNF)

A given relation is called in BCNF if and only if [illegible] already exists in 3NF.

[illegible] for every non-trivial Functional dependency α → β, α is a super key of the relation.

1NF

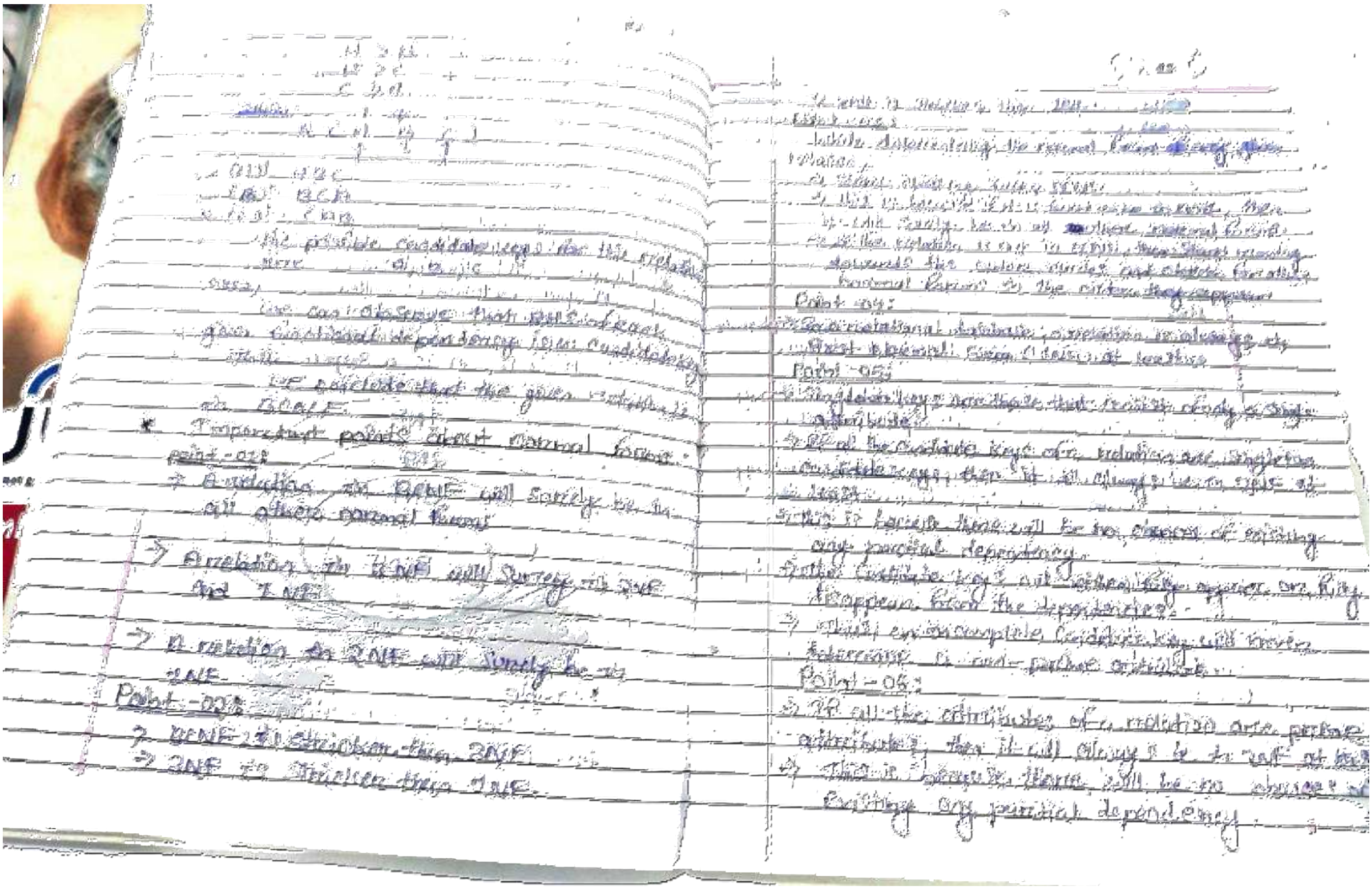
2NF

3NF

BCNF

Example

Consider a relation - R (A , B , C) with the functional dependencies :-



A → B
B → C
C → A

AD: ABC
BD: BCA
CD: CAB

The possible candidate keys for the relation are AD, BD, CD.

We can observe that LHS of each given functional dependency is a candidate key.

We conclude that the given relation is in BCNF.

* Important points about normal forms:

Point-01:

→ A relation in BCNF will surely be in all other normal forms.

→ A relation in 3NF will surely be in 2NF and 1NF.

→ A relation in 2NF will surely be in 1NF.

Point-02:

→ BCNF is stricter than 3NF.
→ 3NF is stricter than 2NF.

[illegible] is stricter than 1NF.

Point-03:

While determining the normal form of any given relation,
→ Start checking from BCNF.
→ If it is found to be in BCNF, then it will surely be in all other normal forms.
→ If the relation is not in BCNF, then start moving towards the other normal forms and check for each normal form in the order they appear.

Point-04:

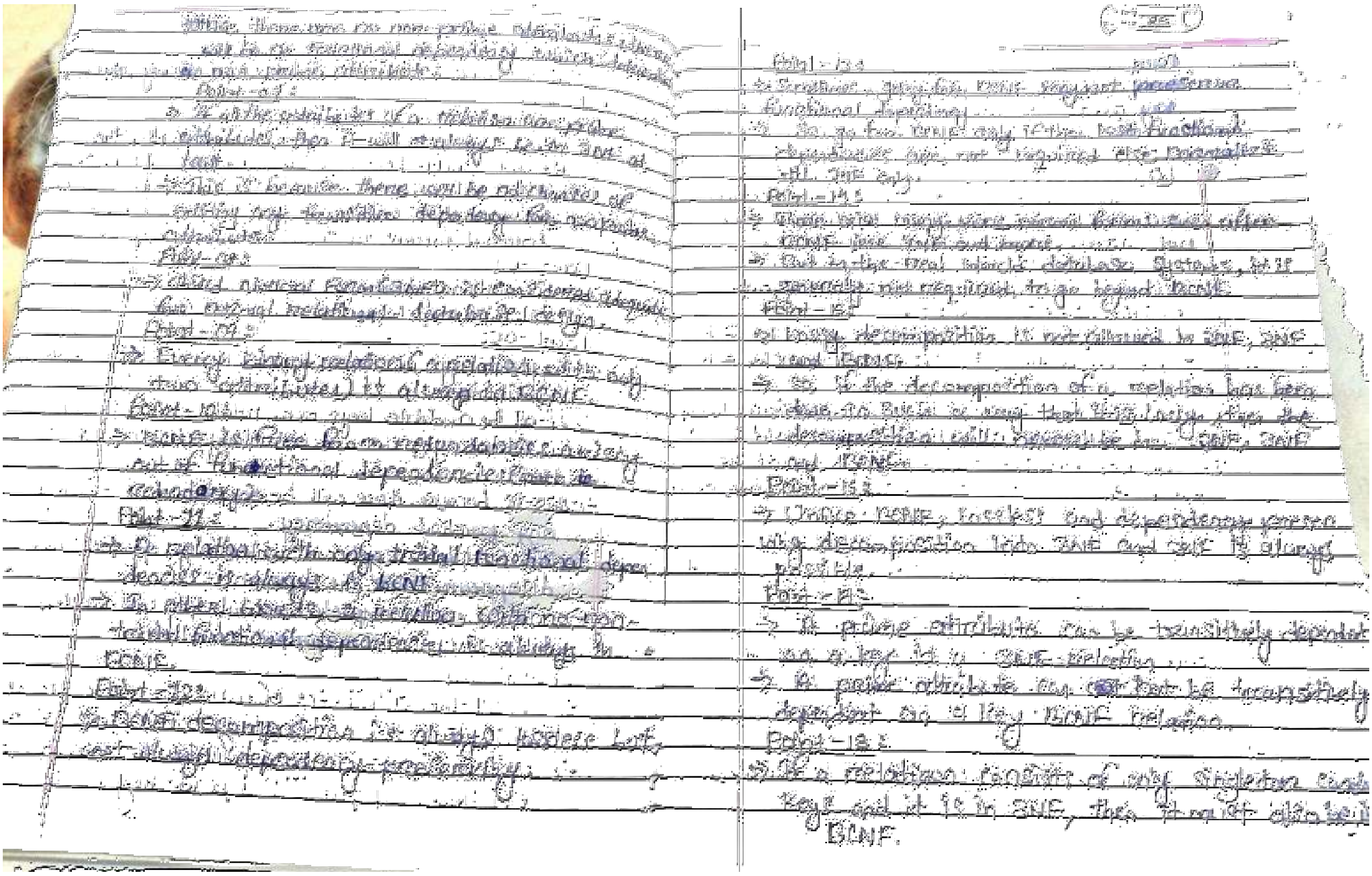
→ In a relational database, a relation is always in first normal form (1NF) at least.

Point-05:

→ Singleton keys are those that consist of only a single attribute.
→ If all the candidate keys of a relation are singleton candidate keys, then it will always be in 2NF at least.
→ This is because there will be no chances of existing any partial dependency.
→ The candidate keys will either fully appear or fully disappear from the dependencies.
→ Thus, an incomplete candidate key will never determine a non-prime attribute.

Point-06:

→ If all the attributes of a relation are prime attributes, then it will always be in 2NF at least.
→ This is because there will be no chances of existing any partial dependency.



Since there are no non-prime attributes, there can be no functional dependency which depends on a non-prime attribute.

Point-07:

⇒ If all the attributes of a relation are prime attributes, then it will always be in 3NF at least.

⇒ This is because there will be no chances of existing any transitive dependency for non-prime attributes.

Point-08:

⇒ Third normal form is considered adequate for normal relational database design.

Point-09:

⇒ Every binary relation (a relation with only two attributes) is always in BCNF.

Point-10:

⇒ BCNF is free from redundancies arising out of functional dependencies (zero redundancy).

Point-11:

⇒ A relation with only trivial functional dependencies is always in BCNF.

⇒ In other words, a relation with no non-trivial functional dependencies is always in BCNF.

Point-12:

⇒ BCNF decomposition is always lossless but not always dependency preserving.

25

Point-13:

⇒ Sometimes, going for BCNF may not preserve functional dependency.

⇒ So, go for BCNF only if the lost functional dependencies are not required else normalize till 3NF only.

Point-14:

⇒ There exist many more normal forms even after BCNF like 4NF and more.

⇒ But in the real world database systems, it is generally not required to go beyond BCNF.

Point-15:

⇒ Lossy decomposition is not allowed in 2NF, 3NF and BCNF.

⇒ So, if the decomposition of a relation has been done in such a way that it is lossy, then the decomposition will never be in 2NF, 3NF and BCNF.

Point-16:

⇒ Unlike BCNF, lossless and dependency preserving decomposition into 3NF and 2NF is always possible.

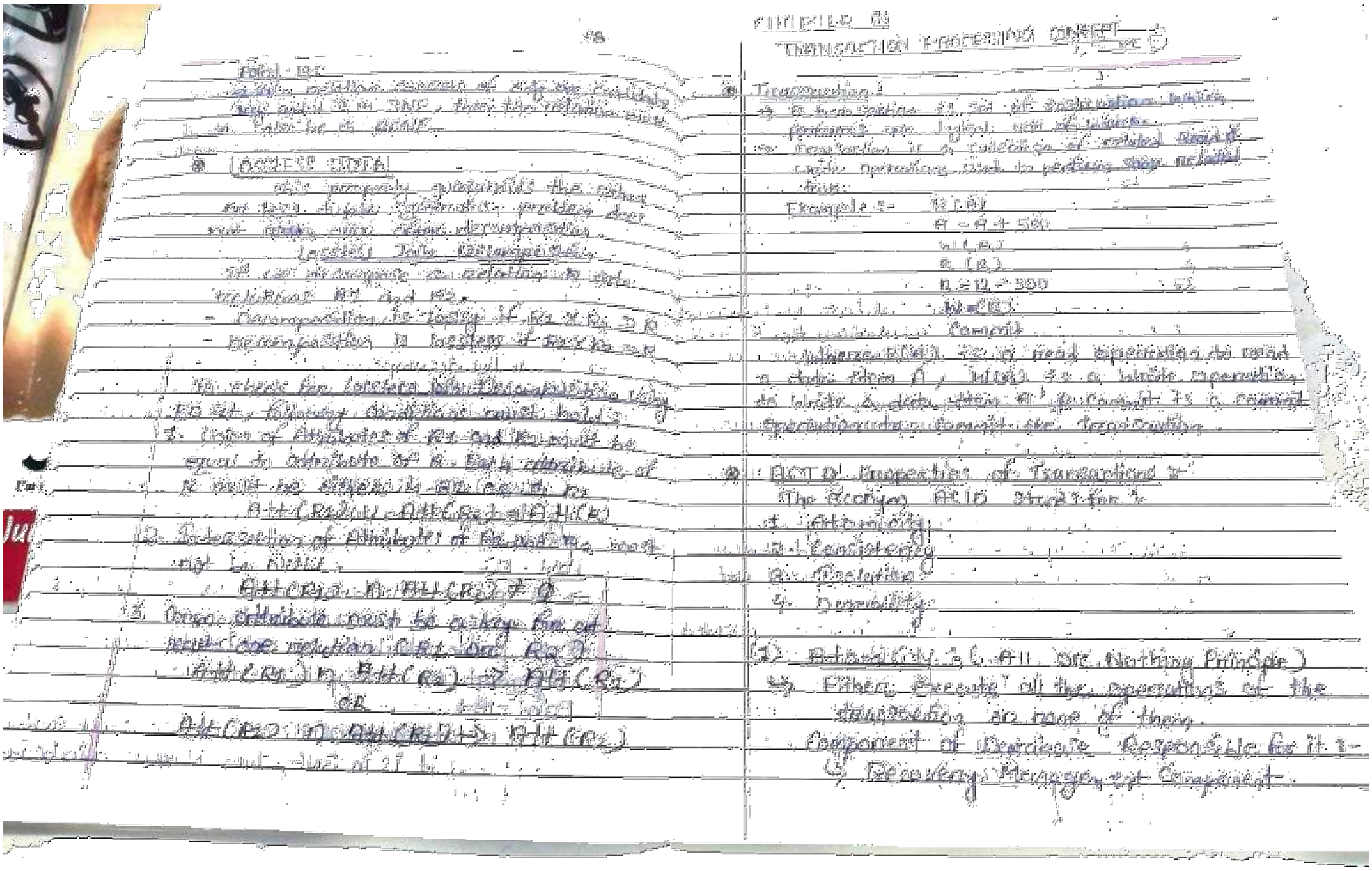
Point-17:

⇒ A prime attribute can be transitively dependent on a key in a 3NF relation.

⇒ A prime attribute can not be transitively dependent on a key in a BCNF relation.

Point-18:

⇒ If a relation consists of only singleton candidate keys and it is in 3NF, then it must also be in BCNF.



Point 1st:

→ [illegible] relation schema of [illegible] key and it is in 3NF, then the relation [illegible] is also be a 3NF.

⊛ Lossless Join

This property guarantees the [illegible] or [illegible] tuple [illegible] problem does not occur after decomposition.

Lossless Join Decomposition.

If we decompose a relation R into relations R1 and R2,

- Decomposition is lossy if $R_1 \bowtie R_2 \supset R$
- Decomposition is lossless if $R_1 \bowtie R_2 = R$

To check for lossless join decomposition using FD set, following conditions must hold:

1. Union of attributes of R1 and R2 must be equal to attributes of R. Each attribute of R must be either in R1 or in R2.

$Att(R_1) \cup Att(R_2) = Att(R)$

2. Intersection of attributes of R1 and R2 must not be NULL.

$Att(R_1) \cap Att(R_2) \neq \emptyset$

3. Common attribute must be a key for at least one relation (R1 or R2)

$Att(R_1) \cap Att(R_2) \rightarrow Att(R_1)$

OR

$Att(R_1) \cap Att(R_2) \rightarrow Att(R_2)$

CHAPTER 05
TRANSACTION PROCESSING CONCEPT

⊛ Transaction:

→ A transaction is set of instructions which performs one logical unit of work.

→ Transaction is a collection of related read & write operations used to perform any related task.

Example:- R(A)
A = A + 500
W(A)
R(B)
B = B - 500
W(B)
Commit

Where R(A) is a read operation to read a data item A, W(A) is a write operation to write a data item A & Commit is a commit operation to commit the transaction.

⊛ ACID Properties of Transactions:

The acronym ACID stands for:

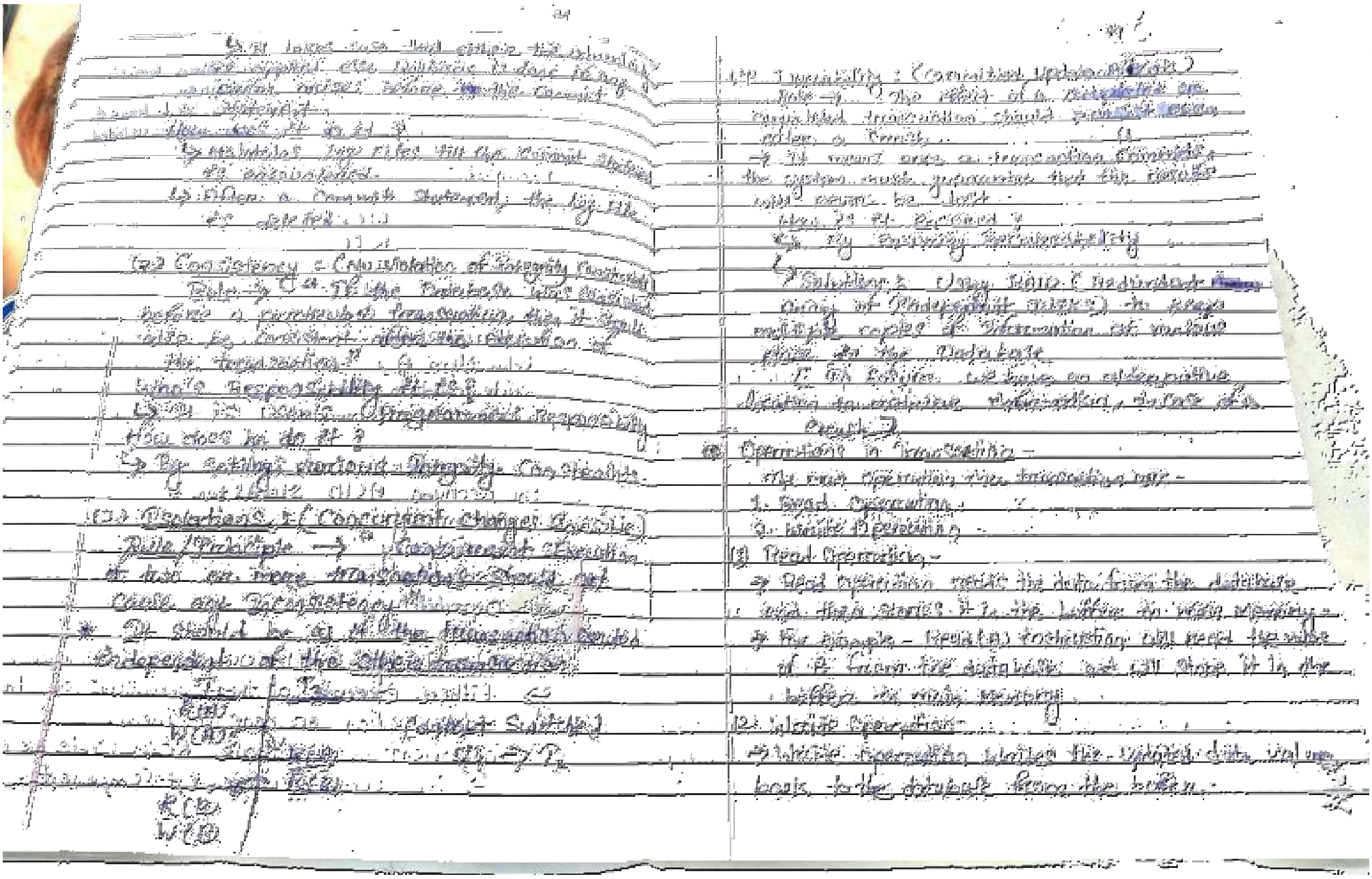
1. Atomicity
2. Consistency
3. Isolation
4. Durability

(1) Atomicity: (All or Nothing Principle)

→ Either execute all the operations of the transaction or none of them.

Component of Database Responsible for it:-

↳ Recovery Manager Component.



↳ It takes care that either the [illegible] [illegible] applied else nothing is done (i.e. no partial [illegible] before the Commit) [illegible].

How does it do it ?

↳ maintains log files till the Commit statement is executed.

↳ When a Commit statement, the log file is deleted.

(2) Consistency :- (No violation of Integrity Constraints)

Rule → "If the Database was consistent before a committed transaction, then it should also be consistent after the execution of the transaction."

Who's Responsibility it is ?

↳ It is users / Programmer's responsibility

How does he do it ?

↳ By setting various Integrity Constraints.

(3) Isolation :- (Concurrent changes invisible)

Rule/Principle → "Concurrent execution of two or more transactions should not cause any Inconsistency [illegible]."

* It should be as if the transactions are independent of the other transactions.

[illegible] (Conflict Serializable)

$T_1 \Rightarrow T_2$

R(A) W(A) [illegible]

R(B) W(B)

(4) Durability : (Committed Updates persist)

Rule → "The effect of a completed or committed transaction should persist even after a crash."

→ It means once a transaction commits, the system must guarantee that the results will never be lost.

How is it ensured ?

↳ By Recovery Responsibility

↳ Solution :- Using RAID (Redundant Array of Independent Disks) to keep multiple copies of information at various place in the Database.

[In Future we have an alternative location to retrieve information, in case of a crash]

Operations in Transaction -

The main operations in a transaction are -

1. Read Operation.
2. Write Operation.

(1) Read Operation -

→ Read operation reads the data from the database and then stores it in the buffer in main memory.

→ For example - Read(A) instruction will read the value of A from the database and will store it in the buffer in main memory.

(2) Write Operation -

→ Write operation writes the updated data value back to the database from the buffer.

→ New changes [illegible] will write the updated values of A from the buffer to the database.

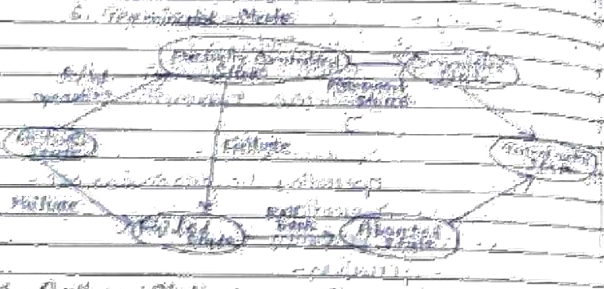
→ Transaction States –

A transaction goes through many different states throughout its life cycle.

These states are called as transaction states.

Transaction states are as follows –

1. Active State
2. Partially Committed State
3. Committed State
4. Failed State
5. Aborted State
6. Terminated State



1. Active State [illegible]

→ This is the first state in the life cycle of transaction [illegible]

→ A transaction is called in an active state as long as its instructions are getting executed.

→ All the changes made by the transaction now are stored in the buffer in main memory.

2. Partially Committed State –

→ After the last instruction of transaction has executed, it enters into a partially committed state.

→ After entering this state, the transaction is considered to be partially committed.

→ It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.

3. Committed State –

→ When all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.

→ Now, the transaction is considered to be fully committed.

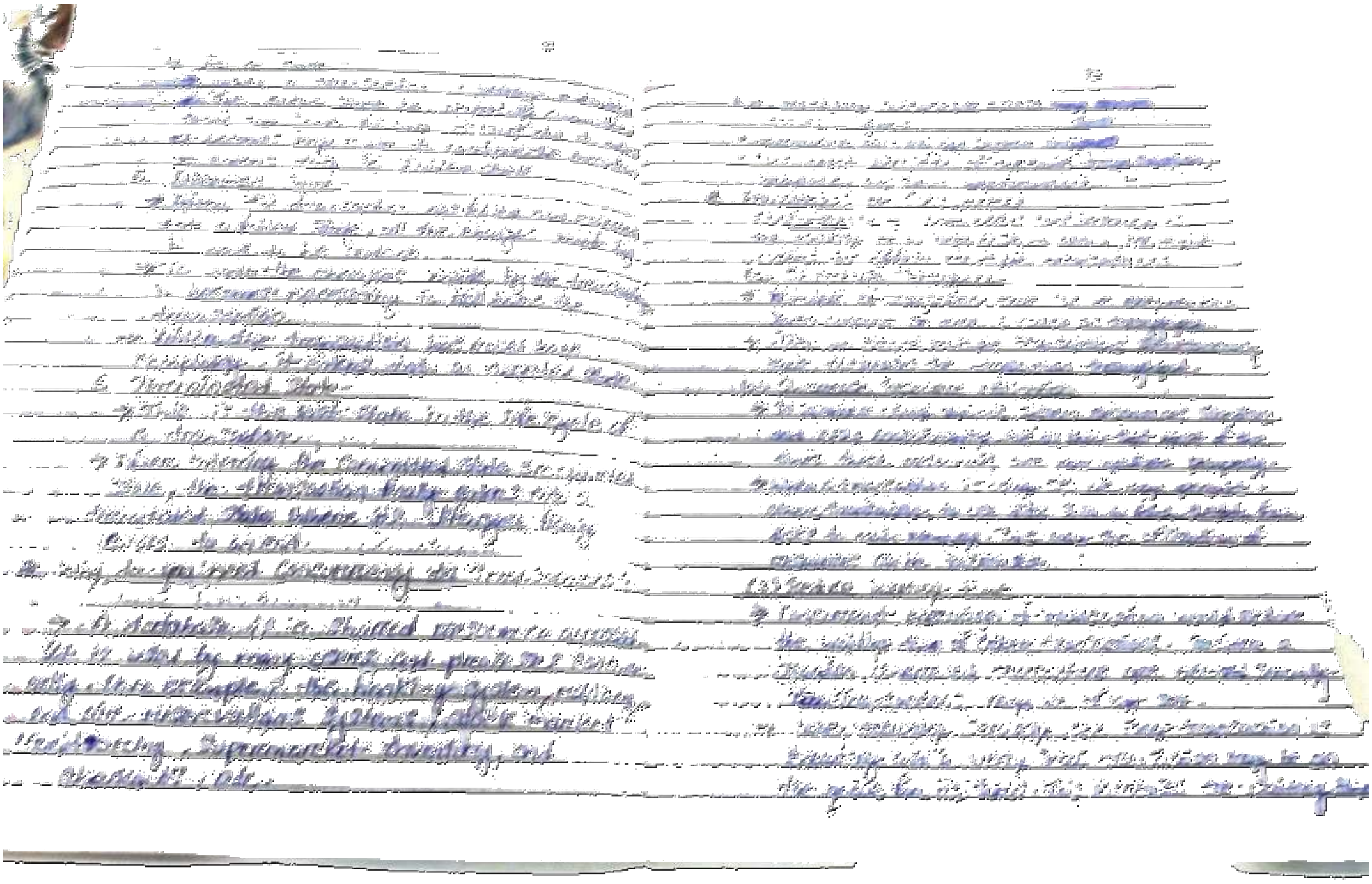
NOTE –

→ After a transaction has entered the committed state, it is not possible to roll back the transaction.

→ In other words, it is not possible to undo the changes that has been made by the transaction.

→ This is because the system is updated into a new consistent state.

→ The only way to undo the changes is by carrying out another transaction called as compensating transaction that performs the reverse operation



[illegible]

D. [illegible] State

[illegible]

E. Terminated State

- This is the last state in the life cycle of a transaction.
- When entering the committed state or aborted state, the transaction [illegible] state [illegible]

Why do we need Concurrency in Transactions?

[illegible]

A database is a shared resource [illegible] It is used by many users and processes concurrently. For example, the banking system, railway and the reservations system, stock market monitoring, supermarket inventory, and checkouts, etc.

[illegible]

(3) Reduce waiting time

[illegible]

transaction. Hence, if we execute transactions simultaneously, the waiting time would be much reduced when compared to the serial execution.

• Improvement Response Time

→ The average time consumed by a transaction to complete since its start is called as the average response time.

→ In concurrent execution, as multiple transactions are running simultaneously by sharing the system resources, the waiting time is reduced which in turn decreases the average response time.

Concurrency Problems in DBMS / Disadvantage of Concurrency

→ When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.

→ Such problems are called as concurrency problems.

1. Dirty Read Problem
2. Unrepeatable Read Problem
3. Lost Update Problem
4. Phantom Read Problem

1. Dirty Read Problem

→ Reading the data written by an uncommitted transaction is called as dirty read.

This read is called as dirty read because—

→ There is always a chance that the uncommitted transaction might roll back later.

→ Thus, uncommitted transaction might

make other transactions read a value that does not even exist.

NOTE-

→ Dirty read does not lead to inconsistency always.

→ It becomes problematic only when the uncommitted transaction fails and roll backs later due to some reason.

Example-

T1	T2
R(A)	
W(A)	
	R(A) // Dirty Read
	W(A)
	Commit
Failure	

Here,

1. T1 reads the value of A.
2. T1 updates the value of A in the buffer.
3. T2 reads the value of A from the buffer.
4. T2 writes the updated the value of A
5. T2 commits.
6. T1 fails in later stages and rolls back.

In this example,

→ T2 reads the dirty value of A written by the uncommitted transaction T1.

→ T1 fails in later stages and roll backs.

→ Thus, the value that T2 read now stands to be incorrect.

→ Therefore, database becomes inconsistent

Unrepeatable Read Problem

→ This problem occurs when a transaction gets to read unrepeated i.e. different of the same variable in its different read operation even when it has not updated its value.

Example

T1	T2
R(X)	
	R(X)
W(X)	
	R(X) // Unrepeated Read

Here,

1. T1 reads the value of X (= 10 say).
2. T2 reads the value of X (= 10).
3. T1 updates the value of X (from 10 to 15 say) in the buffer.
4. T2 again reads the value of X (but = 15)

In this example,

→ T2 gets to read a different value of X in its second reading.

→ T2 wonders how the value of X got changed because according to it, it is running in isolation.

3. Lost Update Problem

→ This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Example

T1	T2
R(A)	
W(A)	
	W(A)
	Commit (T2)
Commit	

Here,

1. T1 reads the value of A (= 10 say).
2. T1 updates the value to A (= 15 say) in the buffer.
3. T2 does blind write A = 25 (write without read) in the buffer.
4. T2 commit.
5. When T1 commits, it writes A = 25 in the database.

In this example

→ T1 writes the over written value of X in the database.

→ Thus update from T1 gets lost.

NOTE -

→ This problem occurs whenever there is a write-write conflict.

→ In write write conflict, there are two writes one by each transaction on same data item without any read in the middle.

2. Unrepeatable Read Problem -

→ This problem occurs when a transaction gets to read unrepeated i.e. different value of the same variable in its different read operations even when it has not updated its value.

Example -

T_1	T_2
R(X)	
	R(X)
W(X)	
	R(X) // Unrepeated Read

Here,

1. T_1 reads the value of X (= 10 say).
2. T_2 reads the value of X (=10).
3. T_1 updates the value of X (from 10 to 15 say) in the buffer.
4. T_2 again reads the value of X (but =15).

In this example,

→ T_2 gets to read a different value of X in the second reading.

→ T_2 wonders how the value of X got changed because according to it, it is running in isolation.

3. Lost Update Problem

→ This problem occurs when multiple transactions execute concurrently and updates from one or more transactions get lost.

Example -

T_1	T_2
R(A)	
W(A)	
	W(A)
	Commit (C)
Commit	

Here,

1. T_1 reads the value of A (= 10 say).
2. T_1 updates the value to A (= 15 say) in the buffer.
3. T_2 does blind write A = 25 (write without read) in the buffer.
4. T_2 commit.
5. When T_1 commits, it writes A = 25 in the database.

In this example,

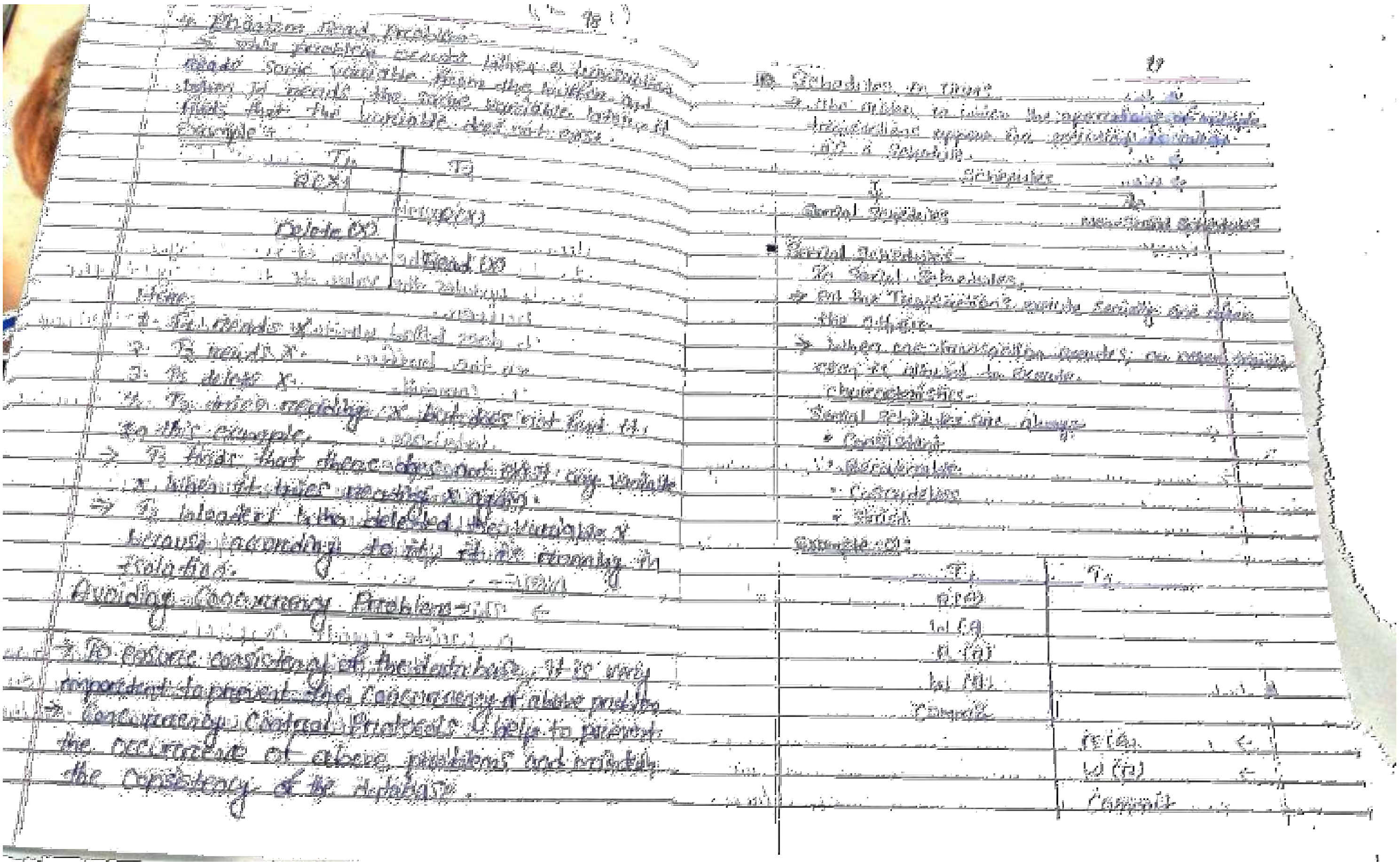
→ T_1 writes the over written value of X to the database.

→ Thus, update from T_1 gets lost.

NOTE -

→ This problem occurs whenever there is a write-write conflict.

→ In write write conflict, there are two writes one by each transaction on same data item without any read in the middle.



18

Phantom Read Problem-

→ This problem occurs when a transaction reads some variable from the buffer and when it reads the same variable later, it finds that the variable does not exist.

Example:-

T1	T2
R(X)	
	Read(X)
Delete(X)	
	Read(X)

Here,

1. T1 reads X.
2. T2 reads X.
3. T1 deletes X.
4. T2 tries reading X but does not find it.

In this example,

→ T2 finds that there does not exist any variable X when it tries reading X again.

→ T2 wonders who deleted the variable X because according to it, it is running in isolation.

Avoiding Concurrency Problems-

→ To ensure consistency of the database, it is very important to prevent the occurrence of above problems.

→ Concurrency Control Protocols help to prevent the occurrence of above problems and maintain the consistency of the database.

17

10. Schedules in DBMS

→ The order in which the operations of multiple transactions appear for execution is called as a schedule.

Schedules

Serial Schedules — Non-Serial Schedules

■ Serial Schedules-

In Serial Schedules,

→ All the transactions execute serially one after the other.

→ When one transaction executes, no other transaction is allowed to execute.

Characteristics-

Serial schedules are always-

- Consistent
- Recoverable
- Cascadeless
- Strict

Example-01:

T1	T2
R(A)	
W(A)	
R(B)	
W(B)	
Commit	
	R(A)
	W(B)
	Commit

In this Schedule-

→ There are two transactions T1 and T2 executing serially one after the other.

→ Transaction T1 executes first.

→ After T1 completes its execution, transaction T2 executes.

→ So, this schedule is an example of a serial schedule.

Example-02:

T1	T2
[illegible]	[illegible]
R(A)	
W(A)	
R(B)	
W(B)	
Commit	

In this Schedule-

→ There are two transactions T1 and T2 executing serially one after the other.

→ Transaction T2 executes first.

→ After T2 completes its execution, transaction T1 executes.

→ So, this schedule is an example of a serial schedule.

Non-Serial Schedules:

In non-serial schedules,

→ Multiple transactions execute concurrently.

→ Operations of all the transactions are interleaved or mixed with each other.

Characteristics-

Non-serial schedules are NOT always-

- Consistent
- Recoverable
- Cascadeless
- Strict

Example-01:

T1	T2
R(A)	
W(A)	
	R(A)
R(B)	
W(B)	
Commit	
	R(B)
	Commit

In this schedule-

→ There are two transactions T1 and T2 executing concurrently.

→ The operations of T1 and T2 are interleaved.

→ So, this schedule is an example of a non-serial schedule.

Example-02:

T1	T2
	R(A)
R(A)	
W(B)	
	R(B)
	Commit
R(B)	
W(B)	
Commit	

In this Schedule,

→ There are two transactions T1 and T2 executing concurrently.

→ The operations of T1 and T2 are interleaved.

→ So, this schedule is an example of a Non-Serial Schedule.

Finding Number of Schedules-

Consider there are n number of transactions T1, T2, T3 ...., Tn with N1, N2, N3 ...., Nn number of operations respectively.

Total Number of Schedules-

Total number of possible schedules (serial + non-serial) is given by-

$$\frac{(N_1 + N_2 + N_3 + \dots + N_n)!}{N_1! \times N_2! \times N_3! \times \dots \times N_n!}$$

Total Number of Serial Schedules-

Total number of serial schedules
= Number of different ways of arranging n transactions
= n!

Total Number of Non-Serial Schedules-

Total number of non-serial schedules
= Total number of schedules – Total number of serial schedules

Recoverable Schedules-

If in a schedule,

→ A transaction performs a dirty read operation from an uncommitted transaction

→ And its commit operation is delayed till the uncommitted transaction either commits or roll backs

then such a schedule is known as a Recoverable Schedule.

Here,

→ The commit operation of the transaction that performs the dirty read is delayed.

→ This ensures that it still has a chance to recover if the uncommitted transaction fails later.

Example-

Consider the following schedule-

Transaction T1	Transaction T2
R(A)	
W(A)	
	R(A) // Dirty Read
	W(A)
Commit	
	Commit // Delayed

(Recoverable Schedule)

Here,

→ T_2 performs a dirty read operation.

→ The commit operation of T_2 is delayed till T_1 commits or rollbacks.

→ T_1 commits later.

→ T_2 is now allowed to commit.

→ In case, T_1 would have failed, T_2 has a chance to recover by rolling back.

Irrecoverable Schedules:

If in a schedule,

→ A transaction performs a dirty read operation from an uncommitted transaction

→ And commits before the transaction from which it has read the value

then such a schedule is known as an Irrecoverable Schedule.

Example

Consider the following schedule:

T_1	T_2
R(A)	
W(A)	
.	R(A) // Dirty Read
.	W(A)
.	Commit
.	
Rollback	

Irrecoverable Schedule

Here,

→ T_2 performs a dirty read operation.

→ T_2 commits before T_1.

→ T_1 fails later and roll backs.

→ The value that T_2 read now stands to be incorrect.

→ T_2 cannot recover since it has already committed.

Checking whether a Schedule is Recoverable or Irrecoverable-

Method-01:

Check whether the given schedule is conflict serializable or not.

→ If the given schedule is conflict serializable, then it is surely recoverable.

→ If the given schedule is not conflict serializable, then it may or may not be recoverable.

Note

- All conflict serializable schedules are recoverable.
- All recoverable schedules may or may not be conflict serializable.

Method-02:

Check if there exists any dirty read operation (Reading from an uncommitted transaction is called as a dirty read).

→ If there does not exist any dirty read operation, then the schedule is surely recoverable.

→ If there exists any dirty read operation, then the schedule may or may not be recoverable.

If there exist a dirty read operation, then follow the following cases -

Case-01:

If the commit operation of the transaction performing the dirty read occurs before the commit or abort operation of the transaction which updated the value, then the schedule is irrecoverable.

Case-02:

If the commit operation of the transaction performing the dirty read is delayed till the commit or abort operation of the transaction which update the value, then the schedule is recoverable.

Rule

- No dirty read means a recoverable schedule.

Cascading Schedule

→ If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Schedule or Cascading Rollback or Cascading Abort.

→ It simply leads to the wastage of CPU time.

Example

T_1	T_2	T_3	T_4

R(A)
W(A)			

	R(A)
	W(A)		

		R(A)
		W(A)	

			R(A)
			W(A)

Failure			

(Cascading Recoverable Schedule)

Here,

- Transaction T_2 depends on transaction T_1.
- Transaction T_3 depends on transaction T_2.
- Transaction T_4 depends on transaction T_3.

In this schedule,

→ The failure of transaction T_1 causes the transaction T_2 to rollback.

→ The rollback of transaction T_2 causes the transaction T_3 to rollback.

→ The rollback of transaction T_3 causes the transaction T_4 to rollback.

Such a rollback is called as a Cascading Rollback.

NOTE -

If the transactions T_2, T_3 and T_4 would have committed before the failure of transaction T_1, then the schedule would have been irrecoverable.

Cascadeless Schedule -

→ If in a schedule, a transaction is not allowed to read a data item until the last transaction written it is committed or aborted, then such a schedule is called as a Cascadeless Schedule.

→ A Schedule is cascadeless when it does not have cascading rollback.

→ Cascadeless Schedule allows only committed read operations.

→ Therefore, it avoids cascading roll and thus saves CPU time.

If there exist a dirty read operation, then follow the following cases -

Case - 01 :

If the commit operation of the transaction performing the dirty read occurs before the commit or abort operation of the transaction which updated the value, then the schedule is irrecoverable.

Case - 02 :

If the commit operation of the transaction performing the dirty read is delayed till the commit or abort operation of the transaction which update the value, then the schedule is recoverable.

Rule

- No dirty read means a recoverable schedule.

Cascading Schedule

→ If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a cascading schedule or cascading rollback or cascading abort.

→ It simply leads to the wastage of CPU time.

Example

T_1	T_2	T_3	T_4

R(A)
W(A)			

	R(A)
	W(A)		

		R(A)
		W(A)	

			R(A)
			W(A)

Failure			

(Cascading Recoverable Schedule)

Here,

- Transaction T_2 depends on transaction T_1.
- Transaction T_3 depends on transaction T_2.
- Transaction T_4 depends on transaction T_3.

In this schedule,

→ The failure of transaction T_1 causes the transaction T_2 to rollback.

→ The rollback of transaction T_2 causes the transaction T_3 to rollback.

→ The rollback of transaction T_3 causes the transaction T_4 to rollback.

Such a rollback is called as a Cascading Rollback.

NOTE -

If the transactions T_2, T_3 and T_4 would have committed before the failure of transaction T_1, then the schedule would have been irrecoverable.

Cascadeless Schedule -

→ If in a schedule, a transaction is not allowed to read a data item until the last transaction written it is committed or aborted, then such a schedule is called as a Cascadeless Schedule.

→ A schedule is cascadeless when it does not have cascading rollback.

→ Cascadeless schedule allows only committed read operations.

→ Therefore, it avoids cascading rollback and thus saves CPU time.

19/09

Example -

T_1	T_2	T_3
R(A)		
W(A)		
Commit		
	R(A)	
	W(A)	
	Commit	
		R(A)
		W(A)
		Commit

(Cascadeless Schedule)

NOTE -

- Cascadeless Schedule allows only committed read operations.
- However, it allows uncommitted write operations.

Example -

T_1	T_2
R(A)	
W(A)	
	W(A) // Uncommitted write
Commit	

(Cascadeless Schedule)

CHAPTER-07

CONCURRENCY CONTROL CONCEPTS

110

* Concurrency Control :-

It is the process of managing simultaneous execution of transactions in a shared database, to ensure the serializability of transactions.

Purpose of Concurrency Control

(i) To enforce Isolation ⇒ { one transaction is not interleaving with another transaction

(ii) To Preserve database consistency

(iii) To resolve read-write and write-write conflicts. { conflicting operations }