DBMS

2020-21

2(b)

Strong entity set can exist independently and has its own primary key.

Example: Bank with attributes like bank_id (PK) and name.

Weak entity set depends on a strong entity set for its existence. It does not have a primary key of its own. It uses a foreign key from the related strong entity along with its own attributes to form a composite key.

Example: Account related to a bank with attributes account_number and balance.

Relationship between weak and strong entity:

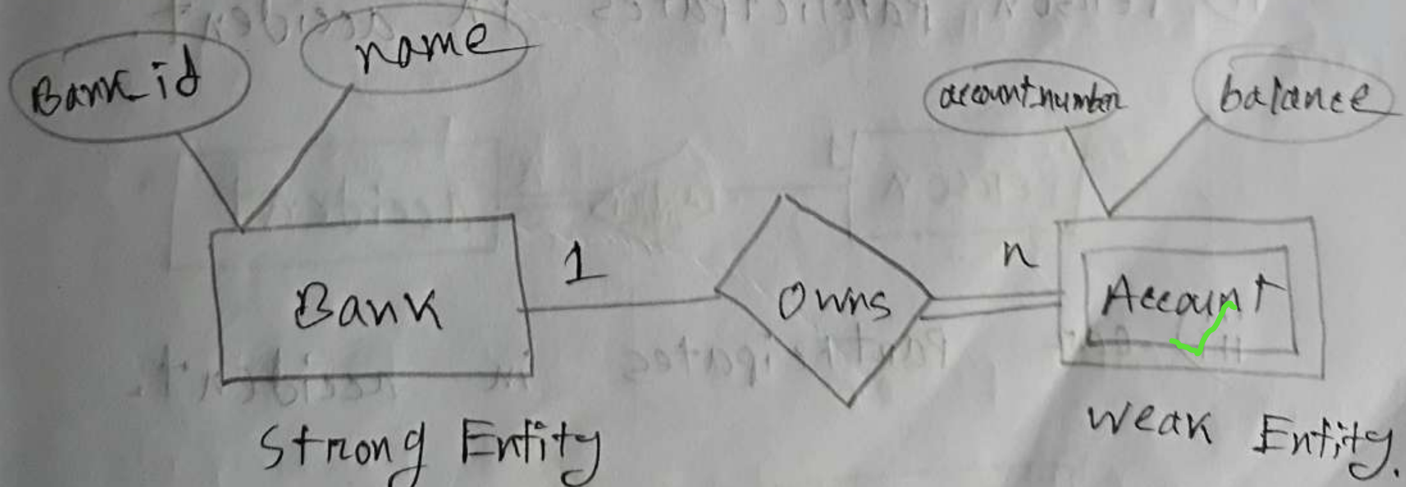
entities: i) Bank (strong entity)
ii) Account (weak entity)

Attributes: Bank: Bank_id, name.

Account: account_number, balance.

Relationships: i) Bank owns account.

ER Diagram:



— x —

2 (d)

Relationships:

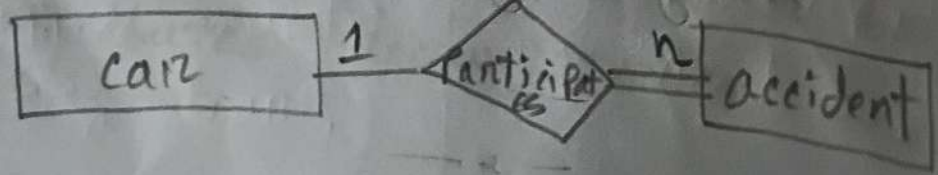
i) person owns car.
ii) person participates in accident.
iii) car participates in accident.

Cardinality ratio and participation:

i) Person owns car.

ii) Person participates in accident.

iii) car participates in accident.



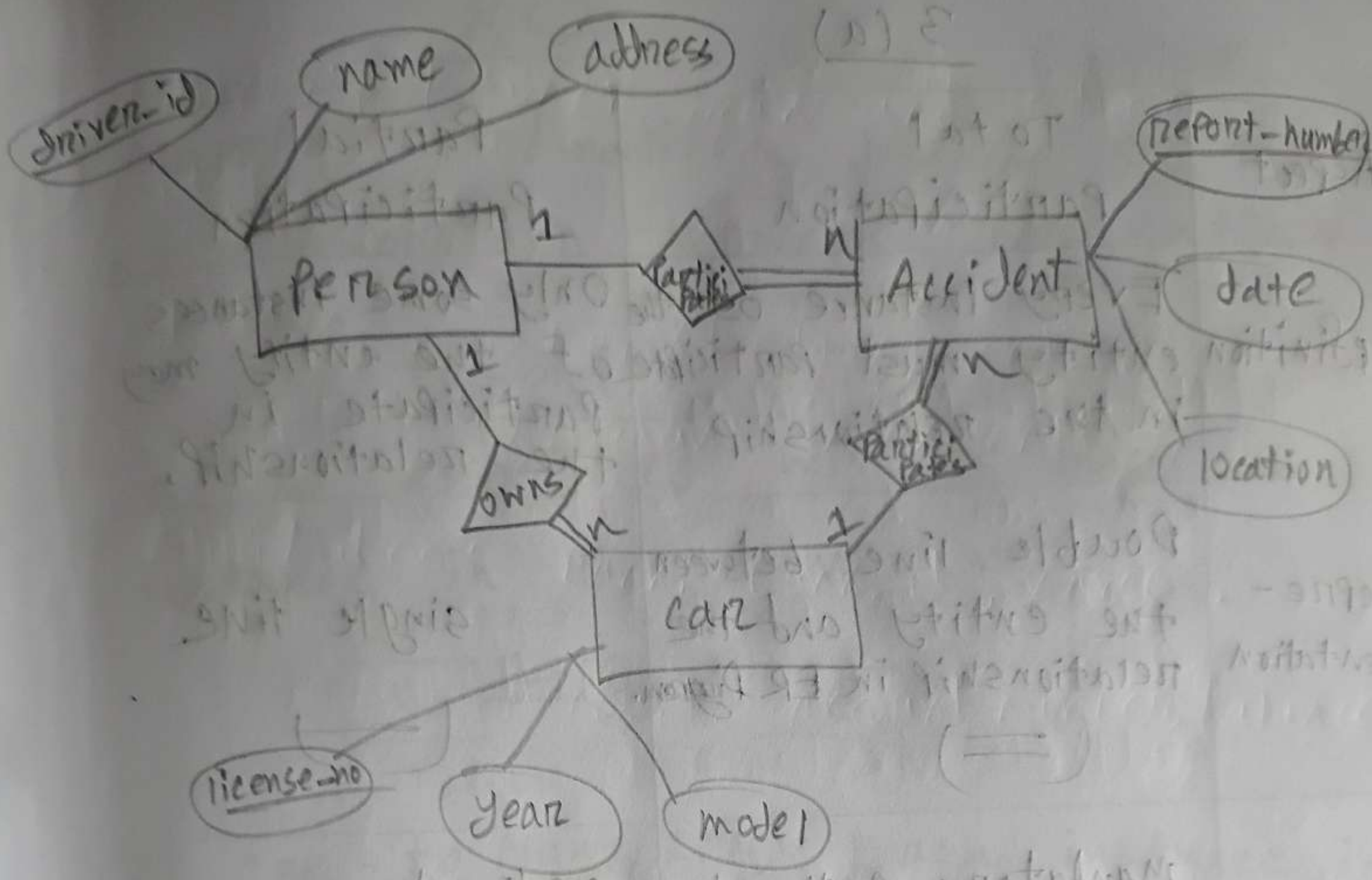


fig: ER Diagram.

— 6 —

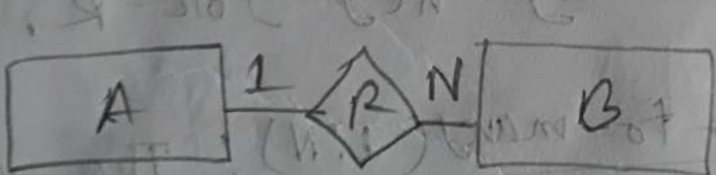
3 (a)

Aspect	Total Participation	Partial Participation
Definition	Every instance of the entity must participate in the relationship	Only some instances of the entity may participate in the relationship.
Representation	Double line between the entity and the relationship in ER Diagram. (═)	single line. (—)
Requirement	Mandatory participation for all instances	optional participation of some instances.
Use case	A "student" must be enrolled in one "course".	A "customer" may or may not have placed an "order".
Example	All "person" entities must own a "car".	Some "person" entities may participate in an "Accident".

i) Mapping cardinalities describe the type of association between two entities.

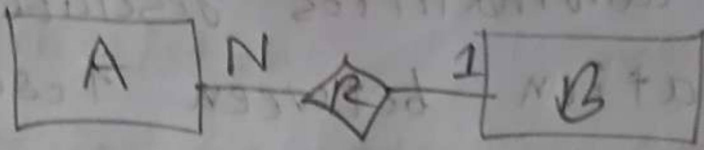
1) One-to-on(1:1): Each instance in A is associated with at most one instance in B, and vice versa.

2) One-to Many(1:N): Each instance in A can be associated with multiple instances in B, but each instance in B is associated with at most one instance in A.

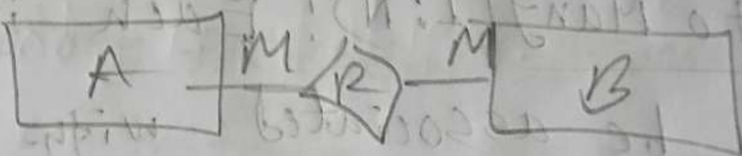


3) Many-to-One(N:1): Each instance in B can be associated with multiple instances in A, but each instance in A is associated with at most one

instance in B.



4) Many-to-Many (M:M): Each instance in A can be associated with multiple instances in B, and each instance in B can be associated with multiple instances in A.



ii)

1) one-to-one (1:1): Either the primary key of A or B can be used as the primary key for R.

2) one-to-many (1:N): The primary key of B and a reference to the primary key of A can define R.

3) Many-to-one (N:1): The primary key of A and a reference to the primary key of B can define R.

4) Many-to-Many (M:M): The primary key for R would be a composite key consisting of bothe the primary keys from A and B.

iii) combining tables can depend on the mapping cardinality:

1) one-to-one: tables A and B can often be combined into a single table, as there is a uniaue pairing.

2) one-to-Many: Include a forign key in the B table referencing A, but it is generally better to keep them separate due to the one-to-many relationship.

3) Many-to-One (N:1) Include a foreign key in the A table referencing B, while keeping them separate.

4) Many-to-Many (M:M): use a separate associative (Junction) table to represent R, with foreign keys referencing both A and B tables.

→

4(a)

Entities

i) Book

ii) Library

iii) Copy

iv) Borrower

v) ~~Loan~~

Attributes

Book: ISBN, Title, Authors.

Library: Name, Type.

Copy: copy-number, Price, loan_date.

Borrower: ID, Name.

~~Loan: Loan-Date.~~

Relationships

i) Library owns copy.

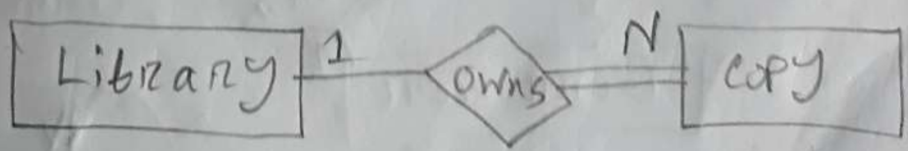
ii) ~~Main~~ Library manages Library.

iii) Borrower borrows a copy of book.

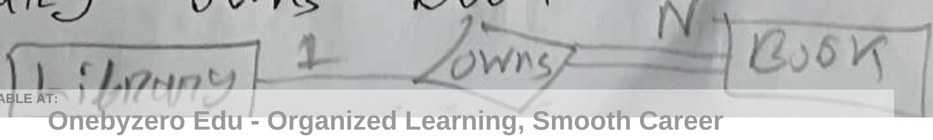
~~iv) Borrower takes loan~~

Cardinality ratio and participation

i) Library owns copy.

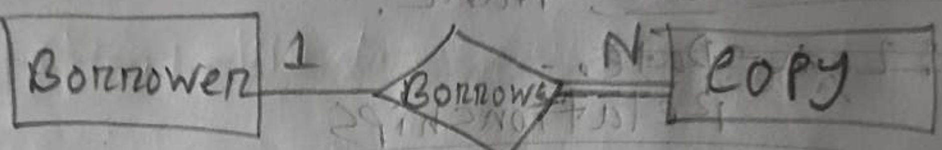


ii) Library owns Book

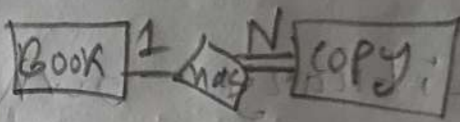


iii) Library manages Library

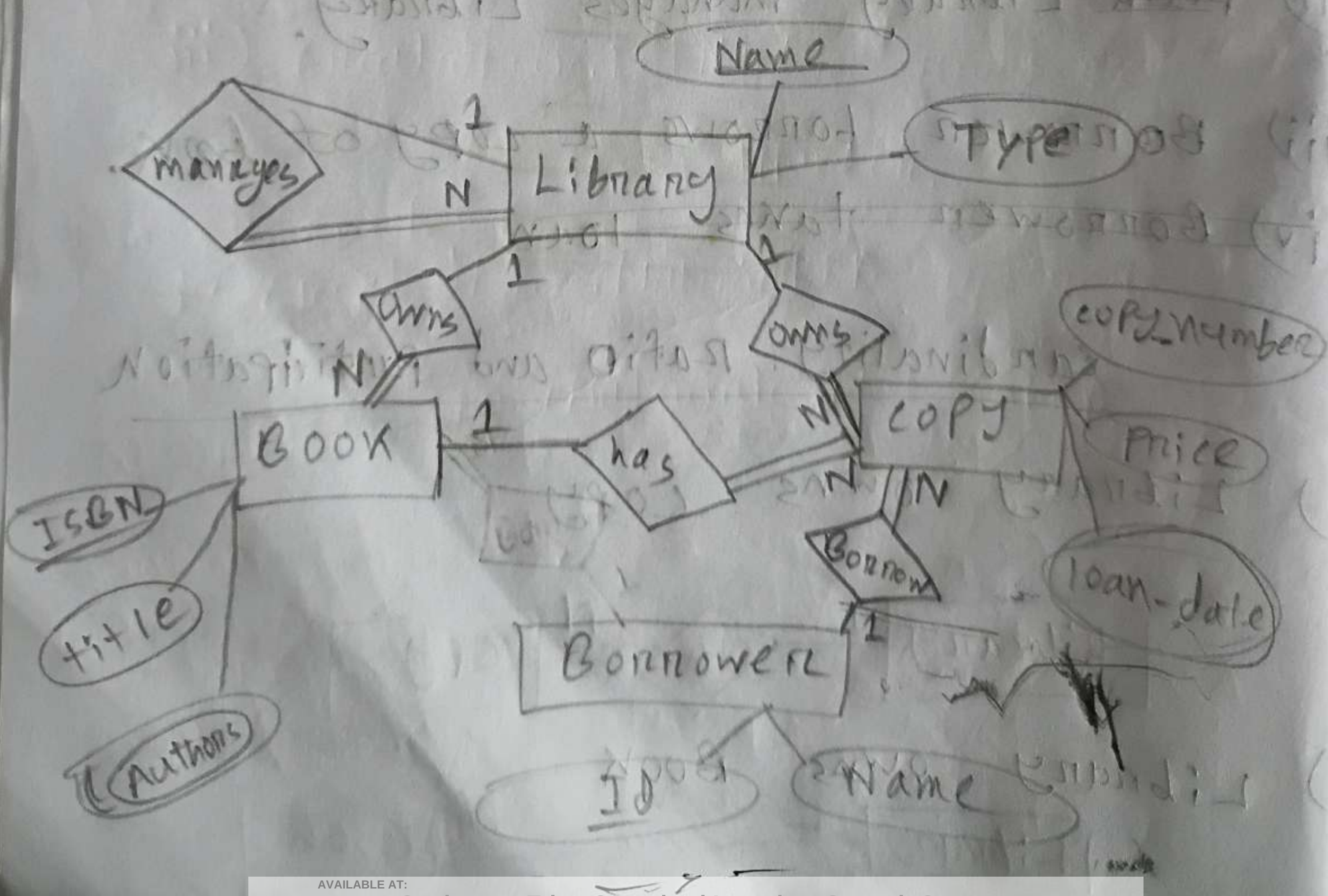
iv) Borrower Borrows Copy



v) Book has copy



E.R Diagram



4(b)

Entities:

i) Lot
ii) ProductionUnits
iii) RawMaterials
~~iv)~~

~~Attributes in~~ Schema:

i) Lot: LotNumber, CreationDate, CostofMateri

ii) ProductionUnits: Unit ID, InstallDate, Producttype.

iii) RawMaterials: MaterialID, Type, Unitcost.

~~Relationship~~

iv) Includes: Lotnumber, Unit ID.

v) created from: LotNumber, Material ID, Units

Referential Integrity constraints:

Primary keys: Each table has a primary key to uniquely identify each row.

Foreign keys:

i) Includes.LotNumber references Lot.Lotnumber.

ii) Includes.UnitID references ProductionUnits.UnitID.

iii) CreatedFrom.LotNumber references Lot.LotNumber.

5(b):[i]
When a tuple in the manager table is deleted, the ON DELETE CASCADE rule ensures that all rows referencing the deleted tuple through the manager_name foreign key are also deleted. This deletion is recursive, that means if those rows are managers for other employees then those rows are also deleted. The cascading continues utill no dependent rows remain. It is like a chain reaction of deletions.

5(b) [ii]:
What will happen if RESTRICT is used instead of cascade?
If RESTRICT is used instead of CASCADE, the deletion of a tuple in the manager table will be prevented if any rows reference the tuple through the manager_name foreign key. This RESTRICT option ensures that a row can't be deleted if it is referenced by another row. It protects referential integrity and preventing accidental deletion.

2019-20

3(a)

Entities

i) Customer

ii) Order

iii) Book

iv) Review

v) Payment.

Attributes

i) Customer: CustomerID, Name, Email, Phone, Address.

ii) Order: OrderID, OrderDate, TotalAmount.

iii) Book: BookID, Title, Author, Prize, Stock.

iv) Review: ReviewID, Rating, ReviewDate.

v) Payment: PaymentID, PaymentDate, Amount, PaymentMethod.

Relationships

i) customer places order.

ii) customer makes payment

iii) customer writes review

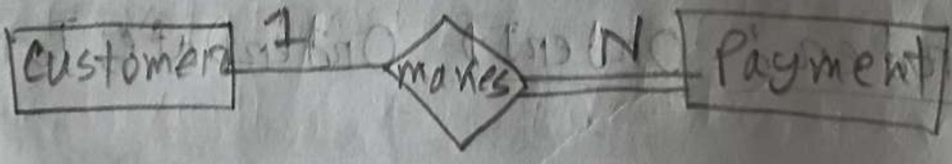
iv) order contains Book

v) Book has review

Cardinality Ratio and ~~Relationship~~ Participation

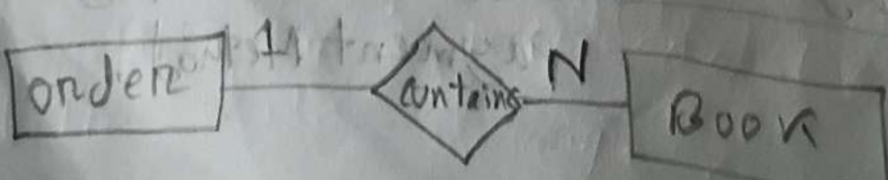
i) customer places order.

ii) customer makes payment.



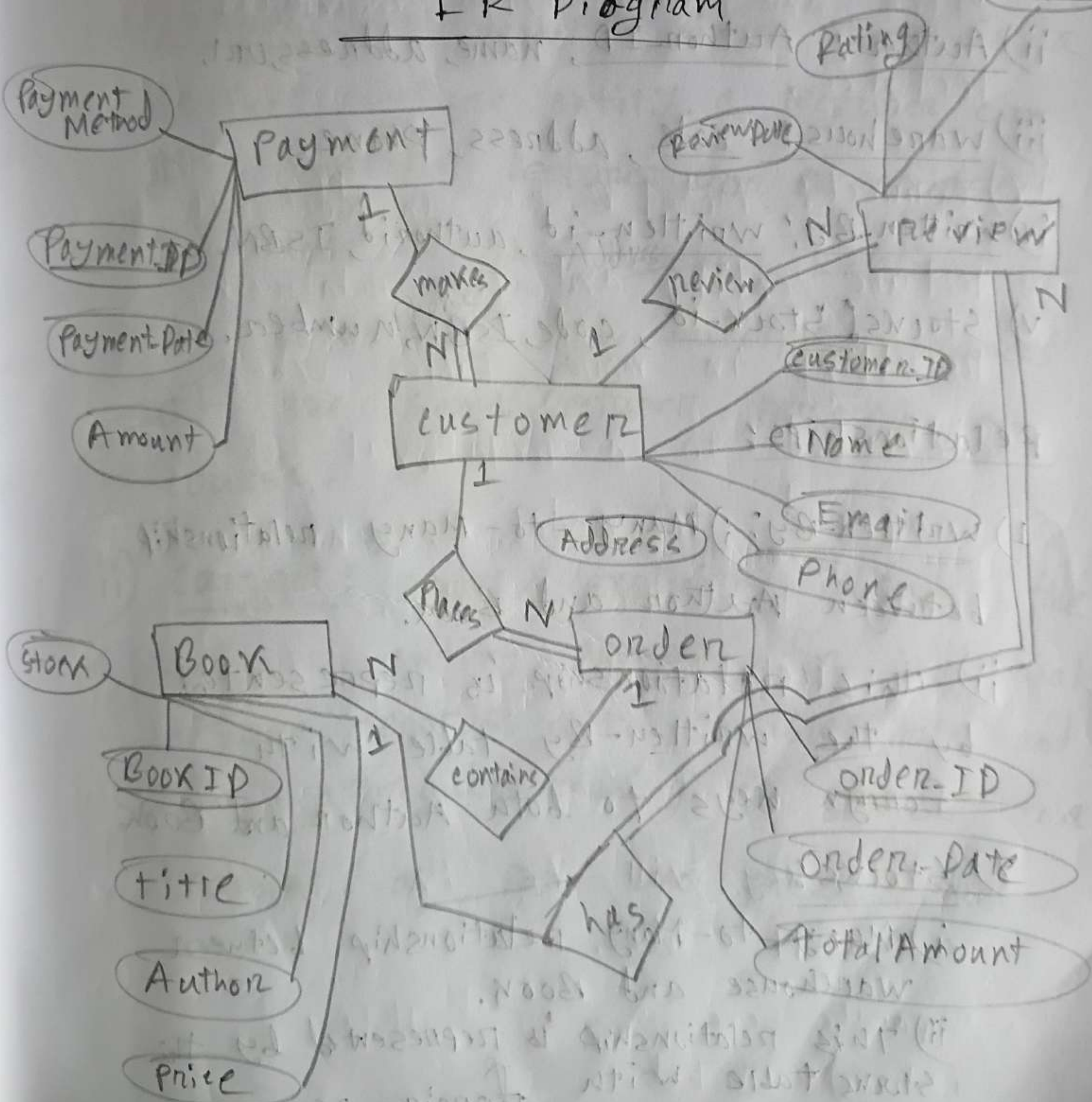
iii) customer writes review

iv) order contains Book



v) Book has review.

ER Diagram



3(c)

Schema for the given Database:

i) Book: ISBN, title, year, Price.

ii) Author: Author_ID, name, address, url.

iii) warehous: code, address, phone.

iv) written-By: written-id, authorid, ISBN.

v) stocks: stock-id, code, ISBN, number.

Relationships:

1) written-By: i) Many-to-Many relationship between Author and Book.
   ii) This relationship is represented by the written-By table with foreign keys to both Author and Book.

2) stocks:
   i) Many-to-Many relationship between warehouse and Book.
   ii) This relationship is represented by the stocks table with foreign keys

to both warehouse and Book.

4(a)

i) Connect: In Model B, since there is a distinct lecture entity, a lecturer can give multiple lectures for a course. In Model A, the teaches relationship can potentially allow multiple instances of the same lecturer teaching a course.

ii) Connect: Model B explicitly includes the lecture entity, allowing for individual lecture records with possible date and time attributes, making it suitable for tracking all past and present lectures.

iii) Correct: Model A lacks a separate lecture entity, so there is no direct way to associate specific dates and times with individual lectures.

iv) Correct: Model B has an additional lecture entity and two relationships, resulting in more tables when converted to a relational schema compared to model A, which only has two entities and on relationship.

---

* Foreign key will be in many / N side.

* Separate schema for double valued attributes.

4(c)

A composite attribute is an attribute that can be broken down into smaller sub-parts, each with independent meaning.

here, createdDate could be a composite attribute because it may contain parts such as day, month and year.

4(d)

Data redundancy occurs when the same piece of information is stored in multiple places, leading to inconsistency.

In this ER diagram, a redundancy issue might exist between Raw Materials and Production Units. Specifically, attributes like Unit of measurement could be duplicated across these entities if they are being managed in both places.

17-18

4(b)

Entities

i) Employee
ii) Department
iii) child

Attributes

i) Employee: ssn, name, salary, phone

ii) Department: dno, dname, budget.

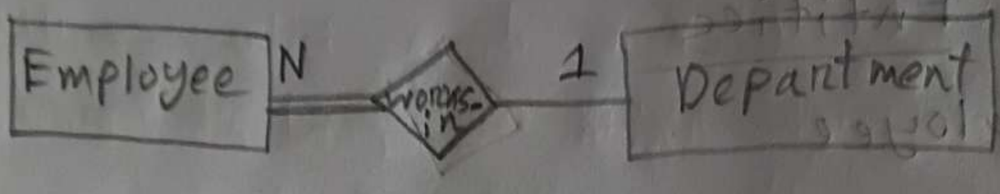
iii) child: name, ssn, age.

Relationship

i) Employee works_in Department.
ii) Employee manages Department.
iii) Employee has child.

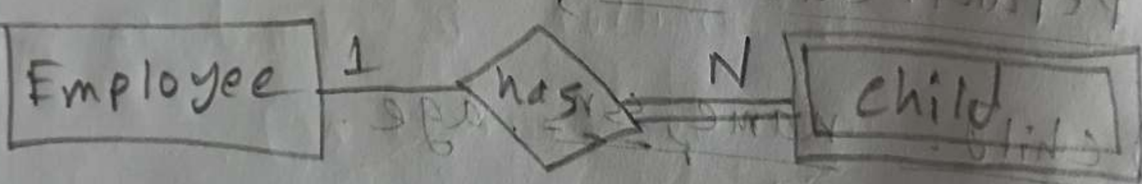
Cardinality Ratio and Participation

i) Employee works_in Department.



ii) Employee manages Department.

iii) Employee has child.



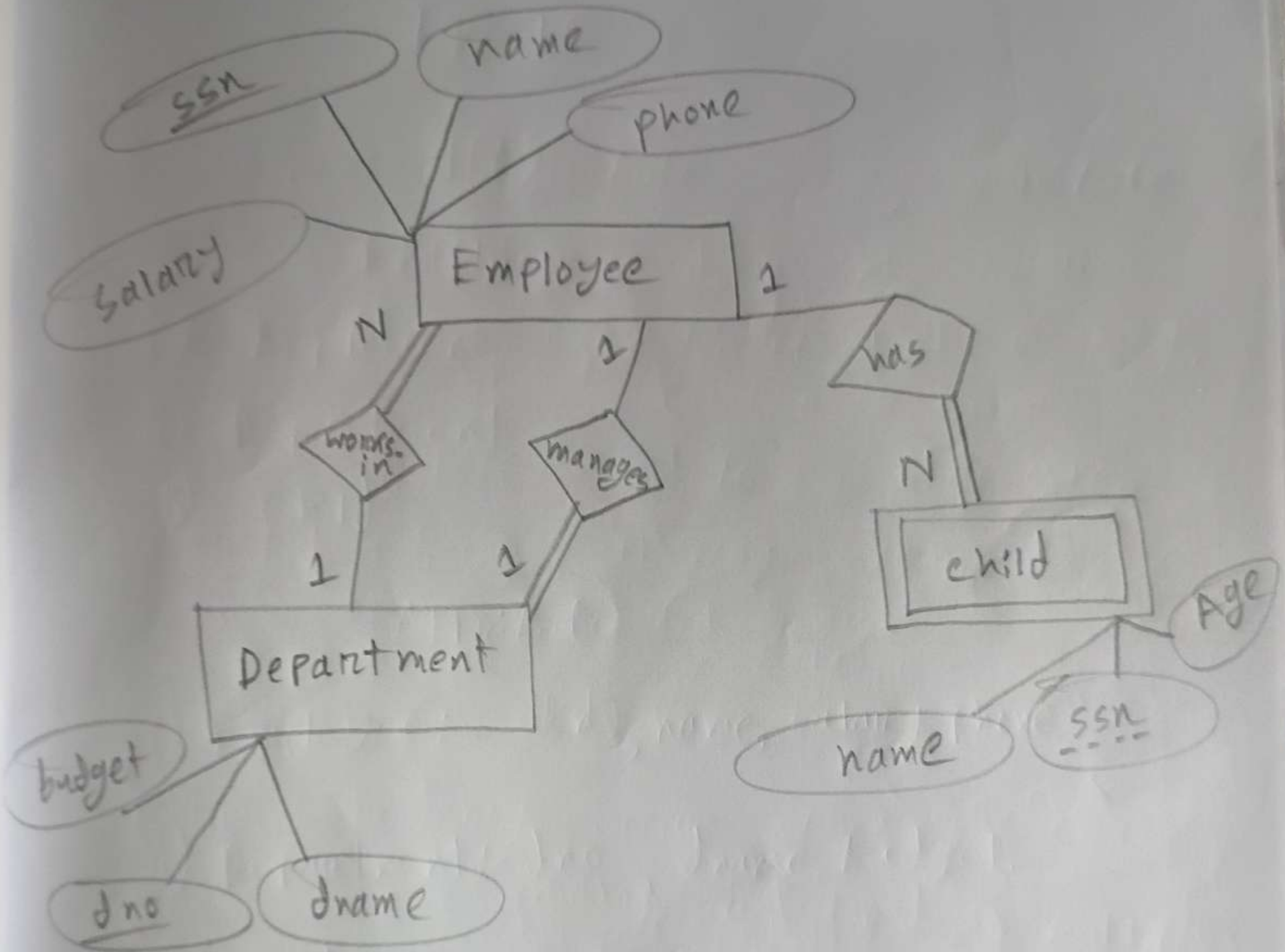


fig: ER Diagram.

CSE-2101

Subject:

Date :

(1) what is database management system (DBMS)? List and explain four reasons why DBMS is used instead of file processing system?

A: Database management system (DBMS) is software that enables the creation, management and manipulation of databases, allowing for efficient data storage and retrieval.

For reasons to use DBMS instead of file processing system:-

i) Reduced Data Redundancy: DBMS centralizes data storage, minimizing duplication across multiple files, which helps maintain consistency.

ii) Enhanced Data Integrity: DBMS enforces data integrity constraints, ensuring that data remains accurate and consistent, whereas file systems may allow errors and anomalies.

Subject:
Date:

iii) Improved Data Access : DBMS supports complex queries and provides efficient data retrieval through languages like, SQL, while file processing systems require more manual coding.

iv) stronger sequrity : DBMS includes robust sequrity features, such as user authentication and access controls, to protect sensitive data, in contrast to the limited security of file processing systems.

② What is the different between a candidate key, a primary key, a composite key, a super key, a Foreign key? What considerations might influence the choice of a primary key?

key Type	Description	Example
candidate key	A set of attributes that can uniquely identify a record. Multiple candidate keys can exist.	EmployeeID, Email

Subject:
Date :

primary key	A selected candidate key that uniquely indentifies records. It must be unique and non-null.	Employee ID (chosen as primary)
composite key	A key that consists of two or more attributes used together to uniquely identify a record	(First Name, Last Name)
super key	A set one or more attributes that can uniquely identify records. It can include extra attributes.	(Employee ID, Email)
Foreign key	An attribute in one table that links to the primary key of another table, establishing a relationship	Department ID in Employee table linking to Department table.

considerations for choosing a primary key:-

i) uniqueness: must uniquely identify each record.

ii) stability: should not change frequently; stable values are preferable.

Subject:
Date :

iii) Simplicity: preferably a single attributes; simpler keys are easier to mange.

iv) Non-nullability: must not allow null values to ensure every record is identifiable.

③ How many attributes you can use a table? Is there any limitations? why you need to split the attributes in multiple tables?

The number of attributes (columns) you can use in a table depends on the DBMS. For example: MS SQL: up to 4096 columns.
postgreSQL: UP to 1,600 columns.
SQL server: UP to 1,024 columns.

Limitations:

i) performance: more attributes can slow down queries and increase resource usage.

ii) <u>complexity</u>: Managing tables with ~~one~~ many attributes can become cumbersome and error-prone.

<u>Reasons to split attributes into multiple tables</u>:

i) <u>Normalization</u>: Reduces data redundancy and enhances data integrity.

ii) <u>Manageability</u>: Smaller tables are easier to understand, maintain, and query.

iii) <u>Efficiency</u>: Improves performance by reducing the amount of data processed during queries.

iv) <u>Flexibility</u>: Easier to manage changes and updates, especially in large datasets.

<u>Extra</u>:

Alternate key	A unique identifier that is not the primary key	Email in an Employees table.

Subject:
Date :

(4) What are the function of DDL and DML in database Language? How they differ from each other?

Functions:

DDL (Data Definition Language):

i) Defines and modifies database structures (e.g., creating, alternating, or dropping tables).

ii) commands includes CREATE, ALTER, DROP, ~~TRUNCATE.~~

DML (Data manipulation Language):

i) Manges and Mainipulates data within those structures.

ii) commands includes S~~ELECT~~, INSERT, UPDATE, DELETE TRUNCATE.

Differences:

i) purpose: DDL focuses on schema design, while DML focuses on data handling.

Subject:
Date :

ii) Impact: DDL changes the database structure, while DML changes the data within the structure.

* Extra:

DQL (Data query Language): SELECT.

TCL (Transaction control Language): COMMIT. Rollback.

DCL (Data control Language): Grant, Revoke.

⑤ Keyword queries in web search are quite different from database queries. List key differences between the two, in terms of way the queries are specified, and in terms of what is the result of a query.

Aspect	Web search queries	Database queries
Specification	Natural language or keywords; less structured.	structured language (e.g., SQL); precise syntax.

Subject:
Date :

Results	Ranked list of relevant web pages; often includes snippets and metadata.	Structured data (tables) with specific fields and records.
Flexibility	Allows for ambiguity and variations in ~~Phar~~ phrasing.	Requires exact matches or defined criteria.
context	contextual relevance based on algorithms (e.g., SEO)	Results based on defined relationships and data integrity.

⑥ what is database trigger? Discuss the strengths and weakness of the trigger mechanism?

A database trigger is a predefined set of instructions that automatically executes in response to specific events (like insertions, updates, or deletions) on a table or view.

Subject:
Date:

Strengths:

i) Automation: Triggers automate repetitive tasks, reducing the need for manual intervation.

ii) Data Integrity: Enforce business rules and maintain data consistency at the database level.

iii) Auditing: Facilitate tracking changes and maintaining historical records for compliance.

Weaknesses:

i) Complexity: Can complicate database architecture, making it harder to mange.

ii) Performance Impact: May introduced overhead, potentially slowing down operations.

iii) Debugging Difficulty: Issues can be challenging to identify and resolve since triggers run automatically in the background.

Subject:

Date :

Key

Reg id	Sec	name	age	Email-id	phone-no
100	A	Farhan	20		
101	A	Hossan	22		
102	B	Rabbi	20		
103	B	Hossan	21		
104	B	Rana	21		
105	A	Nayon	23		

① Candidate key: এই key uniquely ~~identy~~ identify করে। null value থাকতে পারে। but same value থাকতে না। এটি minimal set.

Ex: Reg-id, Email-id, phone-no

(ii) Super key: এটি single বা multiple keys- হতে পারে এবং uniquely identify করে। candidate keys are a subset of super keys. multiple keys চিহ্নিত করতে হলে candidate keys এর সাহায্য নিতেই হবে।

Ex: Reg-id, Email-id, phone-no, (Reg-id + Email-id),

Subject:
Date:

(Reg_id + Email_id + phone_no), (Reg_id + sec), (Email_id + sec + name), (phone_no + sec + name + age), etc.

(iii) <u>primary key</u>: যেটি Not null and must be unique হতে হবে।

<u>Example</u>: Reg_id

SK → CK → PK → FK

(iv) <u>Alternate key</u>: primary key বাদে বাকি সব candidate keys হলো Alternate key.

<u>EX</u>: Email_id, phone_no.

(v)

section-id	sec_name
1	A
2	B
3	C
4	A

ID	name	age	section-id
1	F	20	2
2	h	21	1
3	r	20	1
4	p	22	3

<u>Foreign key</u>: একটি table এর primary key অন্য একটি table এর Refference key হিসেবে থাকলে তাকে Foreign key বলে। FK এর মান null এবং same থাকতে পারে।

Subject:
Date:

কিন্তু Foreign key এর values গুলো আলাদা table এ থাকতে হবে। একই table এ foreign key থাকতে পারে।

Ex: section-id

(vi) Composite Key: যদি table দুই বা ততোধিক primary keys যুক্ত হয়ে composite key তৈরি করে। তাকে composite-primary key বলা হয়

ID	Name	Roll
1	F	101
2	A	102
3	F	103

Ex: ID + Roll ;

(7) Let $R = (A, B, C, D)$, if AB and BD can uniquely identify a tuple in a relation $r(R)$ separately, How many super keys, candidate keys and primary keys are there?

$R = (M, N, P, S, T)$, if MN and NS —

Subject:
Date :

For R = (A, B, C, D):

i) superkeys: multiple (AB, ABC, ABD, BD, BDC, BDA, ABCD)

ii) candidate keys: 2 (AB, BD)

iii) primary keys: 1 (can be either AB or BD)

For R = (M, N, P, S, T):

i) super keys: multiple (MN, MNP, MNS, MNT, MNPS, MNPT, MNSP, MNSPT, NS, NSP, NSM, NST, NSPT, NSMP, NSMT, NSMPT etc)

ii) candidate keys: 2 (MN, NS)

iii) Primary keys: 1 (can be either MN or NS)

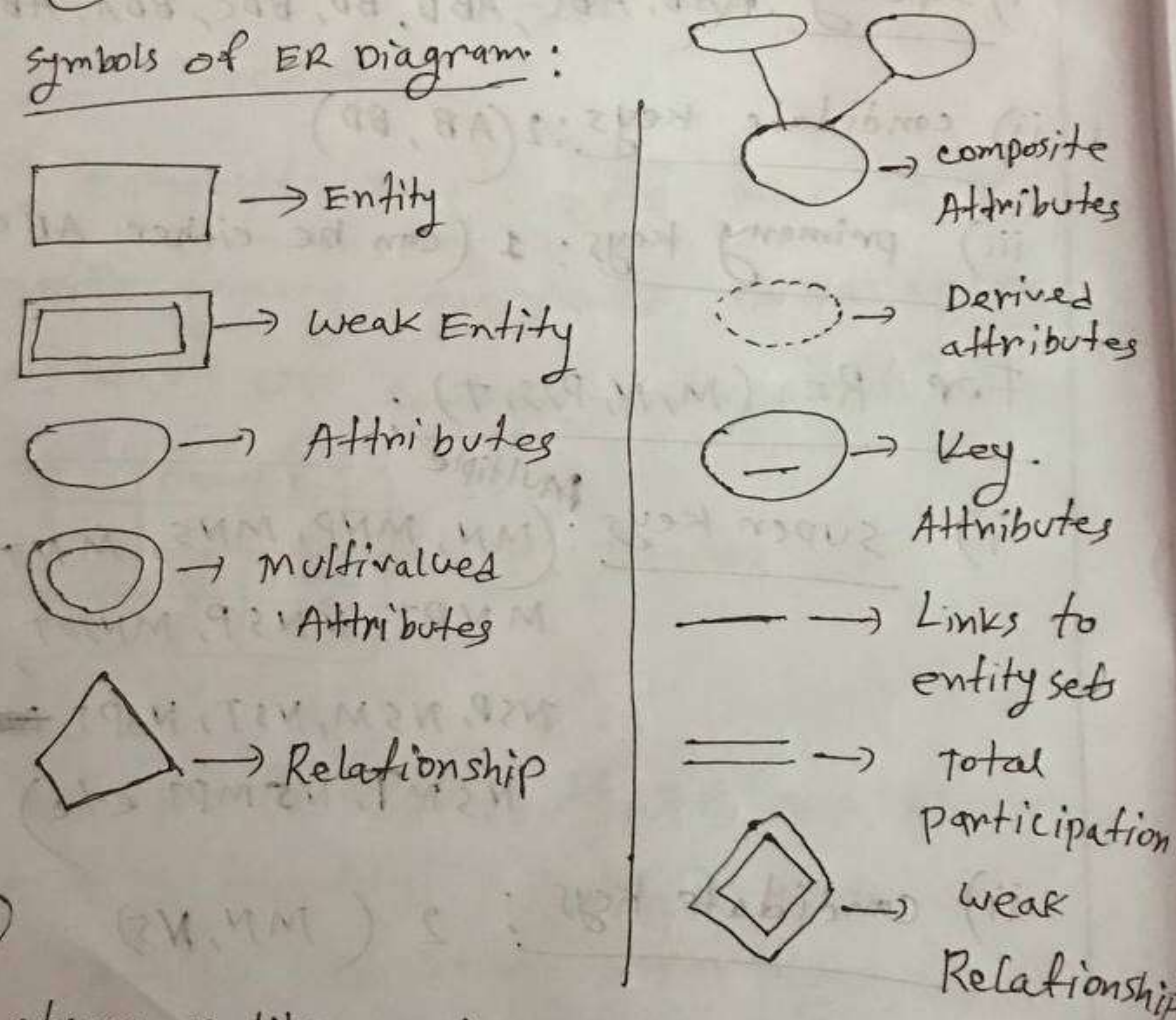
Subject:

Date:

Entity Relationship Diagram

E-R Diagram

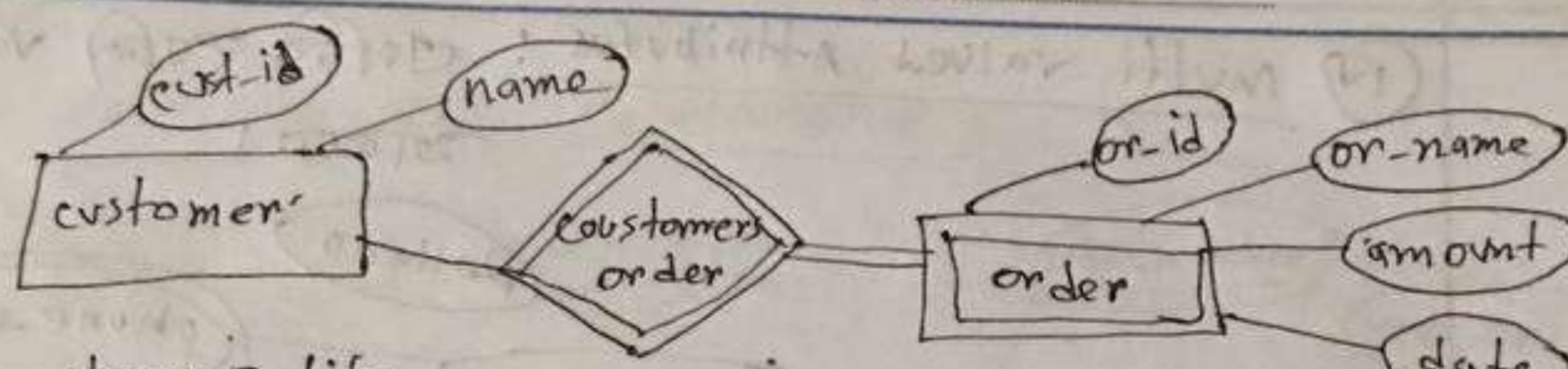
Symbols of ER Diagram:



① <u>Strong Entity</u>: primary key থাকবে।

<u>Weak Entity</u>: primary key থাকবে না তবে। অন্য কোনো Entity এর উপর নির্ভর করবে।

Subject:
Date:

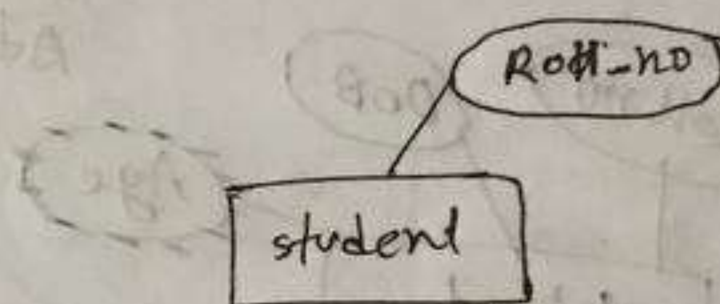


strong Entity

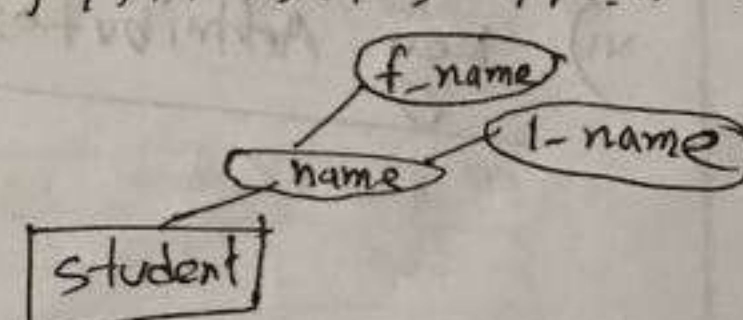
weak Entity

(b)

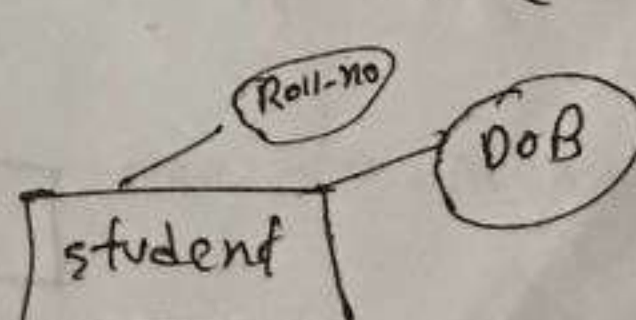
i) Simple Attributes: যে Attributes কে ভাঙলে আর অন্য Attributes পাওয়া যায় না তাকে simple Attributes বলে।



(ii) Composite Attributes: অন্য Attributes গঠিত হয়।

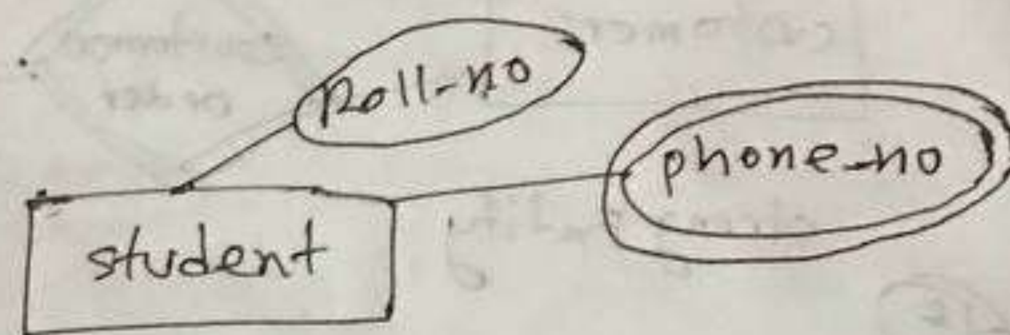


(iii) single-valued Attributes: যেগুলিতে একাধিক value থাকে না

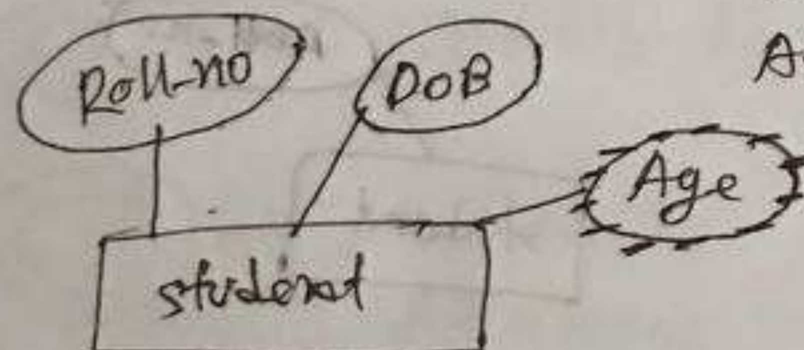


Subject:
Date:

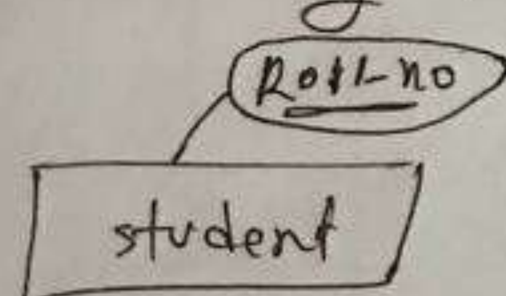
(iv) Multi-valued Attributes: একটির বেশি values থাকবে।



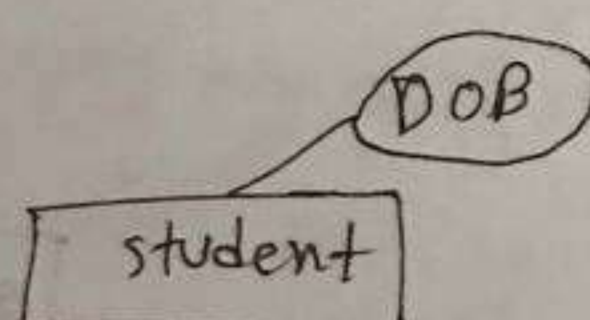
(v) Derived Attributes: যে Attributes এর value অন্য Attributes থেকে নির্ণয় করা যায় সেটিই Derived Attributes.



vi) Key Attributes: primary key.

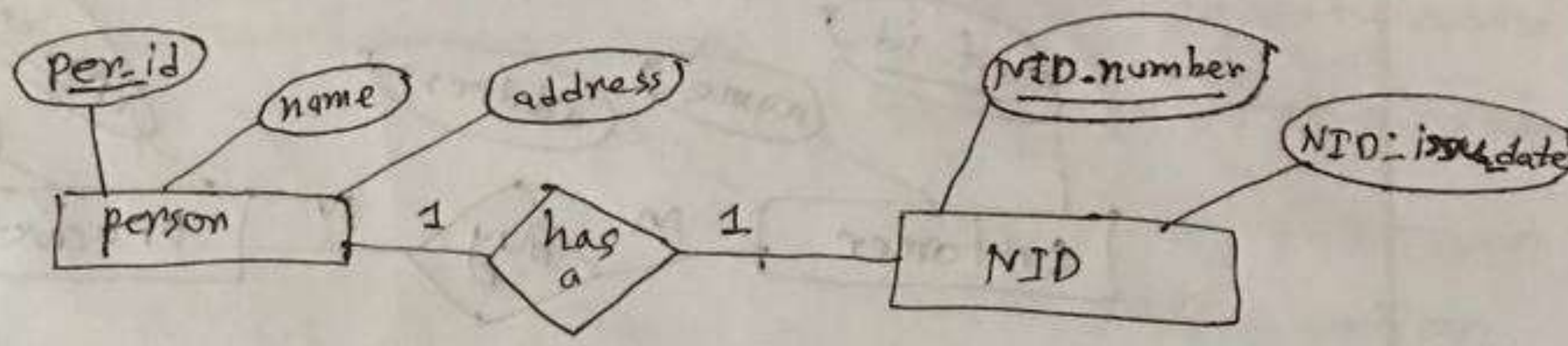


vii) Stored Attributes: যার মান fixed.



Subject:
Date:

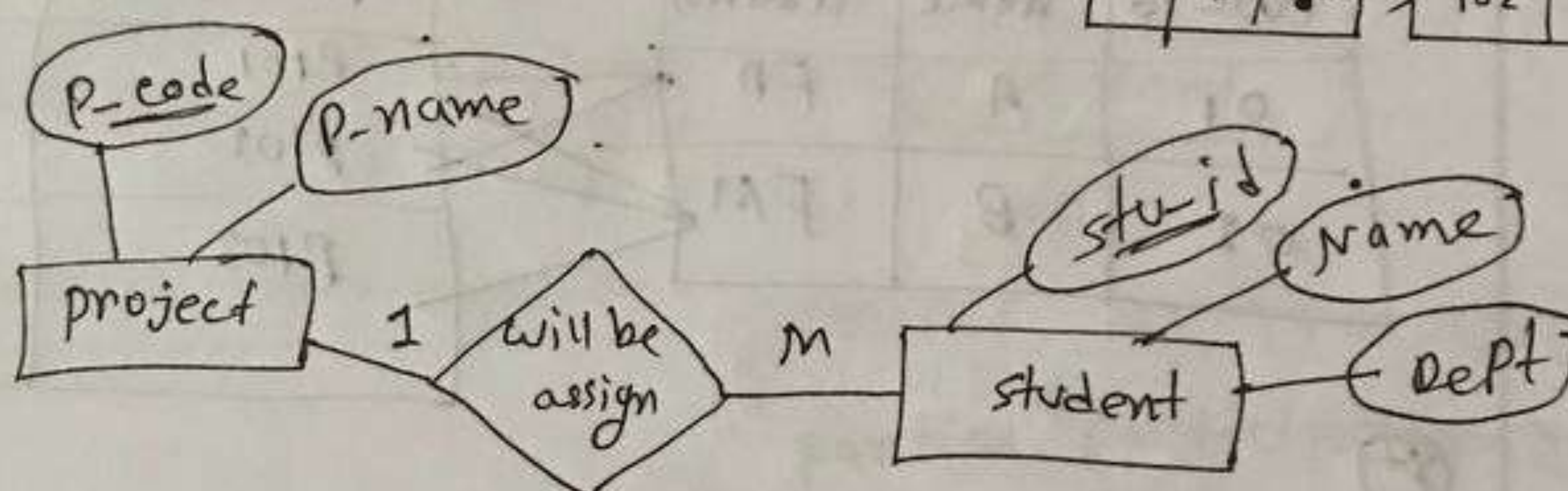
(i) one to one Relationship :



P_id	na	add
01	A	FA
02	B	FB

NID_n	NID_iss
101	25
102	26

(ii) one to Many Relationship :

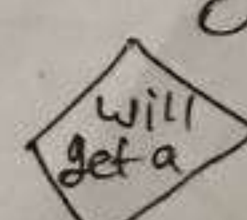


M এর কাছে N দিলেও প্রবলেম নেই।

P_code	P_name
01	A
02	B

stu_id	Name	dept
101	FA	CSE
102	FB	EEE
103	FC	ME
109	FD	CHE

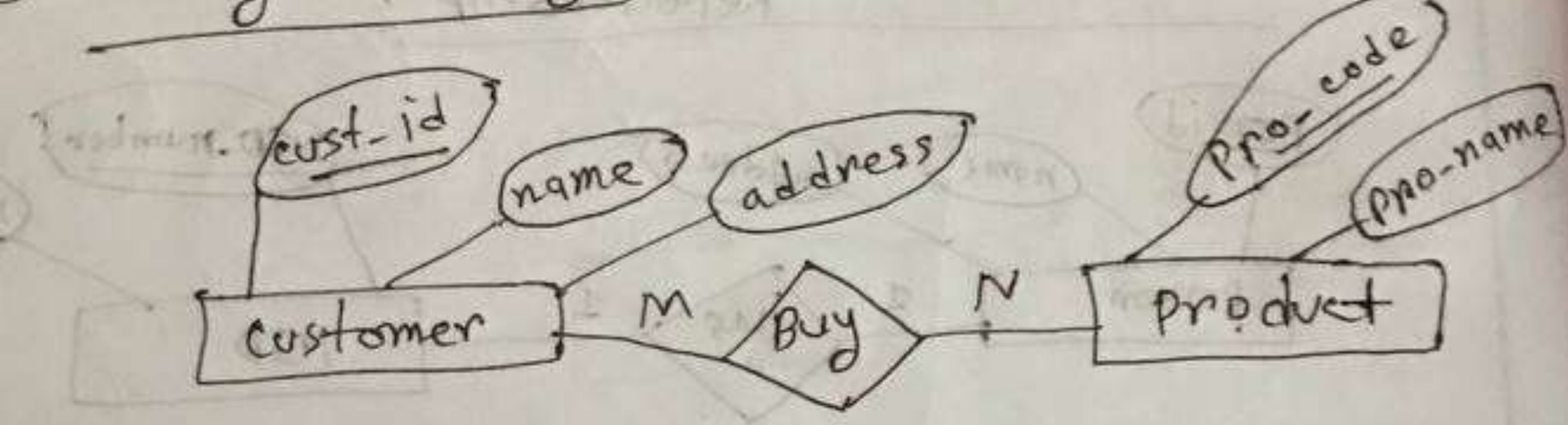
(iii) Many to one : one to many এর উল্টা।



M এর কাছে N দিলেও প্রবলেম নেই।

Subject:
Date :

(iv) Many to many :



cust_id	name	address
01	A	FD
02	B	FM

pro-code	pro-name
P101	AM
P102	BM
P103	CN

(8)

i) Unary : 1 টি Entity থাকলে। Degree 1.

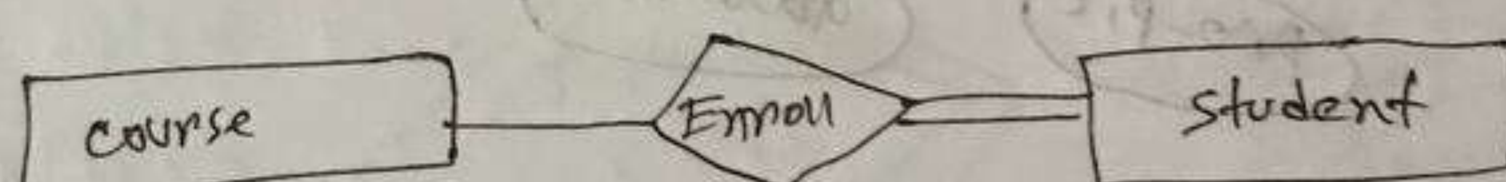
ii) Binary : 2 " " " " 2.

iii) Ternary : 3 " " " " 3.

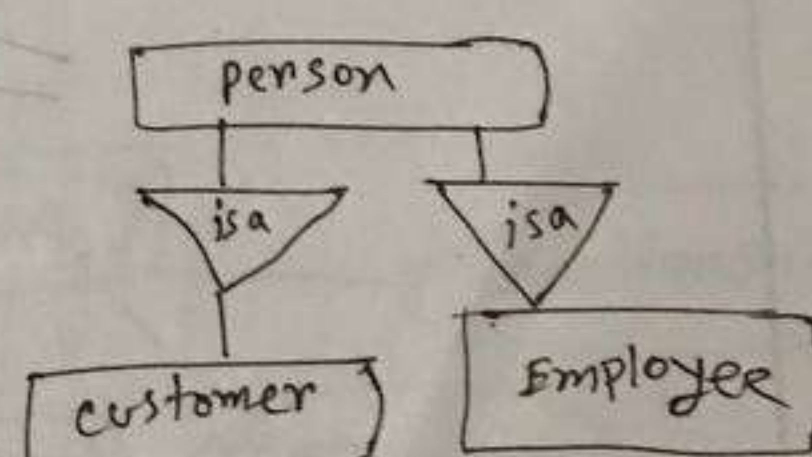
iv) N-ary : n " " " " n.

i) **Total participation:** student entity course entity এর উপর নির্ভরশীল student এর দিক থেকে total participation হবে।

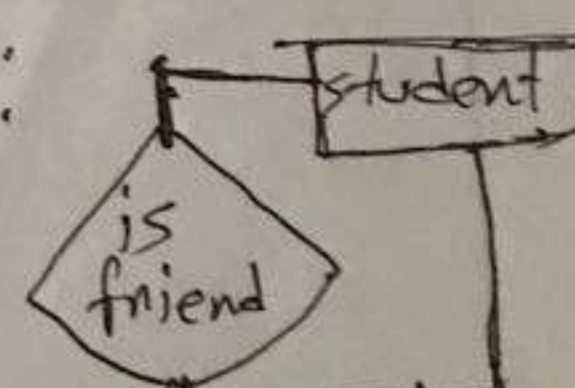
ii) **Partial participation:** course এর দিক থেকে partial participation হবে।



(*) **In a relationship:**

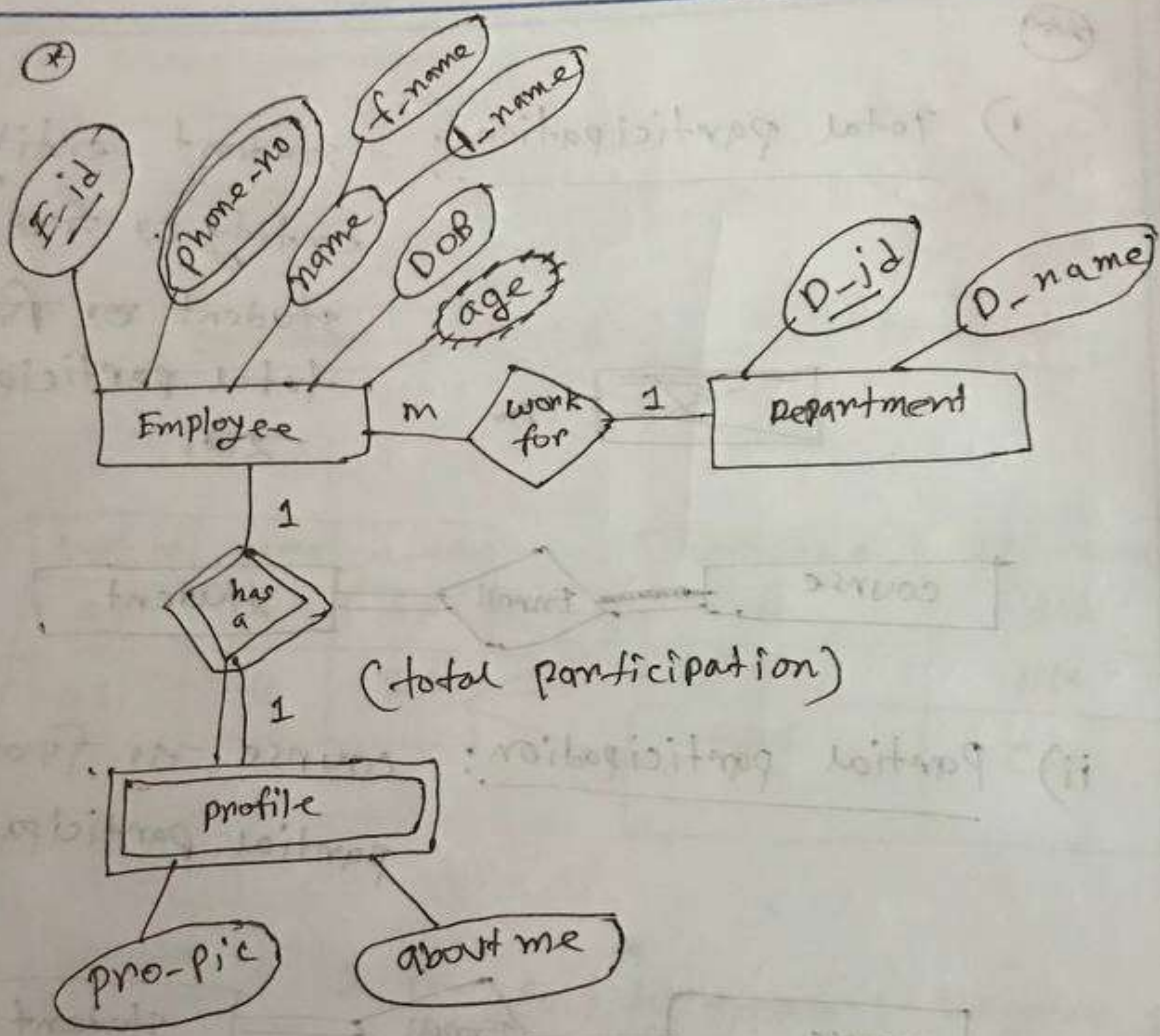


(*) **Recursive / self / unary Relationship:**



Subject:

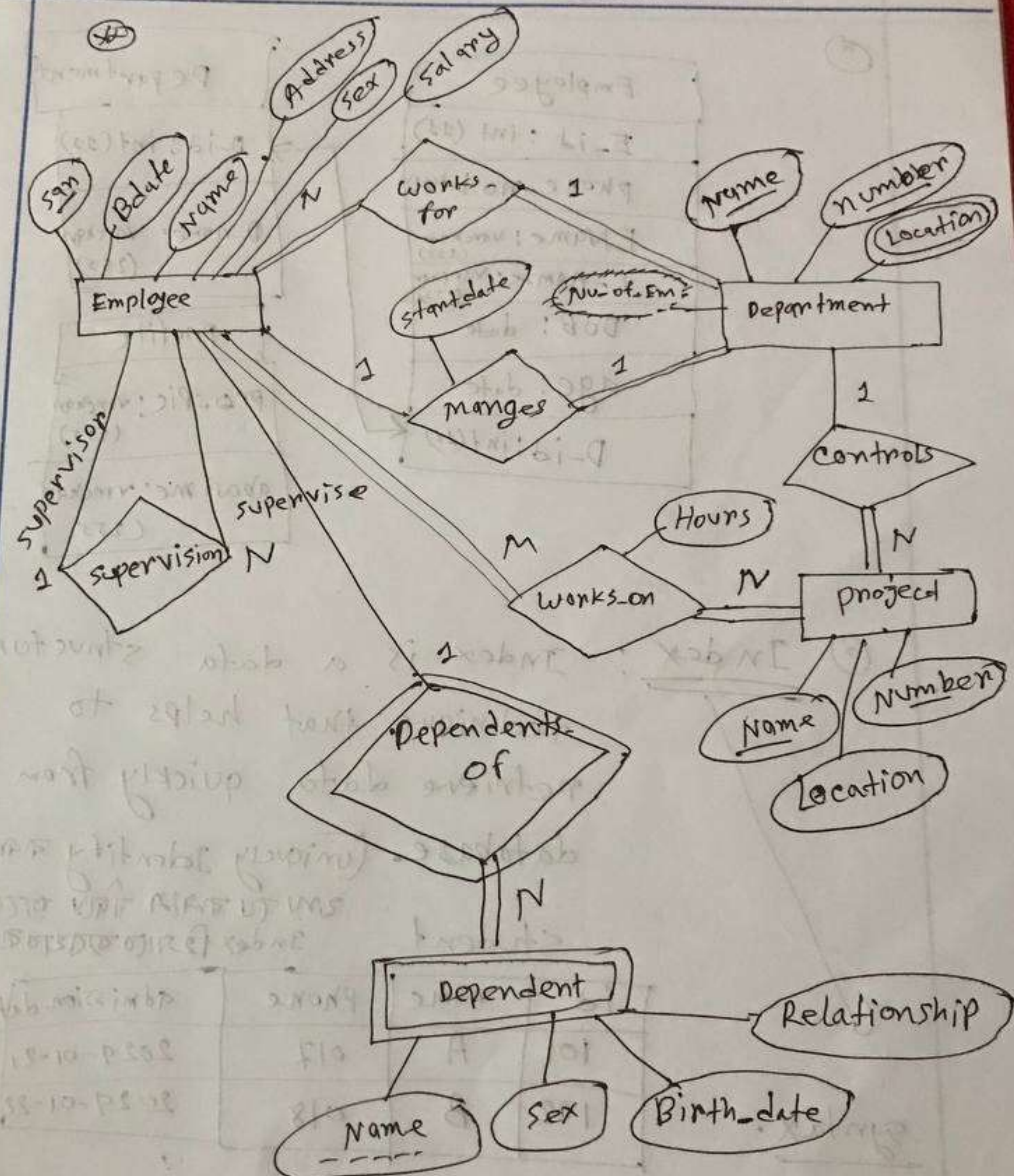
Date :



E-R Diagram

Subject:

Date :



ER Diagram

Functional Dependency (FD)

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

$X \rightarrow Y$

determinant (X) — Dependent (Y)

→ It is a tool for normalization

Advantage

→ Through the identification and removal of redundant or unneeded data.

→ By guaranteeing that data is correct and consistent.

* Describe. Data is dependent on FD,

FD is not dependent on Data.

explain:

FD can exist indepedently, while data cannot exist or be meaningful without FD.

imagin FD is a rule or Function and Data is input.

→ FD: A Function $y = 2x + 3$

→ Date: the value of x or input

FD has meaning the relation between x an y but the data (1, 2, 3) are meaningless.

FD is independent.

Data is dependent.

FD

Trivial

$A \to A$

$AB \to BA$

$B \to B$

if B is subset of A

Non-trivial

$AB \to ABz$

$A \to B$

$B \to C$

B is not subset of A

x, y is a set of attributes

$x \to y$

determinate (x), dependent (y)

R

x	y
1	1
2 (t_1)	1
3	2
4	3
2	6
2 (t_2)	5, 1

$x = 2$ $y = 1$

? must be 1

if $t_1 x = t_2 x$

then $t_1 y = t_2 y$

Example

R.No	Name	marks	Dept	course
1	a	78	CS	c1
2	b	60	ECE	c1
3	a	78	CS	c2
4	b	60	ECE	c3

FD: $x \rightarrow y$

if $t_1 x = t_2 x$
then $t_1 y = t_2 y$

R.No → Name ✓

Name → R.No ✗

R.No → marks ✓

Dept → course ✗

course → Dept ✗

marks → Dept ✓

(R.No, name) → marks ✓

name → marks ✓

(name, mark) → dept ✓

(Name, marks) → (dept, course) ✗

Closure of a Set of Functional Dependencies

It means the complete set of all possible attributes that can be functionally derived from given functional dependency using the Inference rules known as Armstrong's Rule.

→ It denoted by F^+

Inference Rule (IR)

① Reflexive Rule

if $y \subseteq x$ then $x \rightarrow y$

$x = \{a, b, c, d, e\}$

$y = \{a, b, c\}$

② Augmentation Rule

if $x \rightarrow y$ then $xz \rightarrow yz$

For R(ABCD)

if $A \rightarrow B$ then $AC \rightarrow BC$

(iii) Transitive Rule

if $x \to y$ and $y \to z$ then $x \to z$

(iv) Union Rule

if $x \to y$ and $x \to z$ then $x \to yz$

(v) Decomposition Rule

if $x \to yz$ then $x \to y$ and $x \to z$

closure set

$X^+ \to$ contains set of attributes determined by a

Exercise

R(ABCDEFG)

$A \to B$

$BC \to DE$

$AEG \to G$

$(AC)^+ = \{A, C, B, D, E\}$

R(ABCDE)

- A → BC
- CD → E
- B → D
- E → A

$B^+ = \{B, D\}$

$(AB)^+ = \{A, B, C, D, E\}$

B

R(ABCDEFGH)

- A → BC
- CD → E
- E → C
- D → AEH
- ABH → BD
- DH → BC

$(BCD)^+ = \{B, C, D, E, A, H\}$

Irreducable set of FD (canonical cover)

F^c — if F^c don't have
- (i) redundant attribute
- (ii) redundent FD.

Steps :-

1. ___
2. ___
3. ___

F : { AB → C, C → AB, B → C, ABC → AC, A → C, AC → B }

S-1 = { AB → C, C → A, C → B, B → C, ABC → A, ABC → C, A → C, AC → B }

redundant

S-2 = { B → C, C → A, C → B, B → C, A → C, A → B }

S-3 = { C → A, ~~A → C~~, A → B }, B → C }

?

A = { A, B, C }

B = { B

C = { C, B }

C = { C, A, B }

redundant

C = { C, A, B }

B = { B }

A = { A, B }

C = { C, B }

A { A C

Find Key

super keys : set of attributes whose closure contain of all attributes of given Relation

candidate key

candidate keys are super keys with the least number od attributes.

- Any CK can not be proper subset of Any other super keys

Like $A^+ = \{A, B, C, D\} \longrightarrow$ SK, CK

$AD^+ = \{A, B, C, D\} \longrightarrow$ SK but not CK

$\{A\}$ $\{D\}$

R(ABCD)

FD: {A → B, B → C, C → A}

$ABCD^+ = \{A, B, C, D\}$ → SK

$ACD^+ = \{A, C, D, B\}$ → SK

$AD^+ = \{A, D, B, C\}$ → SK

$\{A\}^+ = \{A, B, C, A\}$

$\{D\}^+ = \{D\}$

CK: AD, BD, CD

Prime Attribute

{A, D}

R(ABCDEF)

FD: {AB → C, C → D, B → AE}

AB → D

$ABCDEF^+ = \{A, B, C, D, E, F\}$ → SK

ABDEF = {A, B, D, E, F, C} → SK

ABEF = {A, B, F, C, D, E} → SK

BF = {B, A, E, C, D, F} → SK

(?)

R = (ABCDEF)   C → D, C → E

F = { AB → C, C → DE, E → F, D → A, C → B }

AB → DE

$ABCDEF^+$ = {A, B, C, D, E, F}

$ABDEF^+$ = {A, B, D, E, F, C}

ABF^+ = {A B F C DE}

AB = {A, B, C, D, E, F} → CK.

{A^+} = {A}

{B} = {B}

AB → C
C → E
E → F
∴ AB → F

must | non must
F

Prime attribute

{A, B, D, C}

AB
↓
D B → CK

{D^+} = {D, A}

{B} = {B}

A C → not candidate k.

C^+ = {DE C A, B C} → CK

CK ⇒ AB, DB, C

NORmalization

(Anomaly)

NORmalization is a technique to reduce data redundancy from a table.

* what is data redundancy ?

→ Repetition of similar data at multiple places.

* Repetition of data increases the size of database so we reduce redundancy.

others problem

* Insertion problem
* Deletion
* Updation

Student table.

Roll	name	age	dept	hod
01	A	22	CSE	ME
02	B	23	CSE	ME
03	C	~~24~~ 22	CSE	ME
04	D	23	CSE	ME

To insert redundant data for every new row is a data insertion ANOMALY

Now delete Row one by one delete student info. also dept. info.

- Loss of related dataset when some other dataset is deleted called deletion Anomaly

- Updation Anomaly

Student table info

Student table

Roll	Age	name	dept
01	22	A	CSE
02	23	B	CSE

dept. table

dept	name
CSE	ME

- Insertion
- updation
- deletion

(A) 1st NF

→ Table should not contain any multivalued Attribute

Student table

Roll	Name	course
1	Abir	C++, Java
2	Kabir	DBMS
3	Nibir	C+, DSA

solution : each attribute of a table must have atomic (single) values.

(P1)

Roll	Name	course
1	Abir	C++
1	Abir	Java
2	Kabir	DBMS
3	Nibir	C++
3	Nibir	DSA

(OR)

(P2)

Roll	Name
1	Abir
2	Kabir
3	Nibir

Roll	course
1	C++
1	Java
2	DBMS
3	C++
3	DSA

Roll + course

composit key

8, 256,

2NF

HP elit book 840 G3

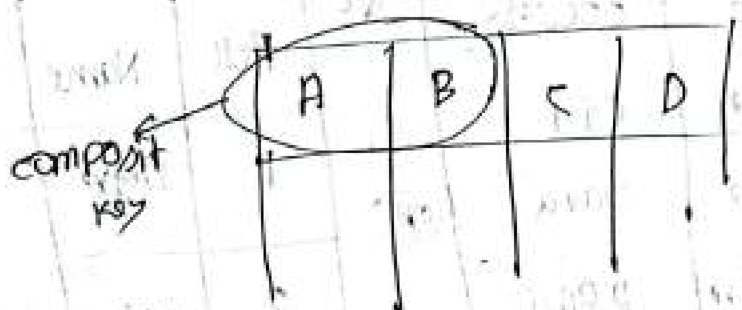
→ Table is in 1NF

→ can't partial dependency

250
10
10
60
50
20
10
50

⊛ partial dependency

Partial dependency is a situation in which a non-key attribute of a table depends on only a part of the primary key.



Std Id	Course Id	course name	Inst.
1	101	Math	X
1	102	Eng	Y
2	101	His	Z

AB → D

B → D

Here (Std Id + cours Id) is composit primary key.

course name → Std Id

cours name depends on part of the primary key Std Id.

How to remove partial dependency

Std

Stud-id	course-id	course fee
1	c1	1000
2	c2	1500
1	c4	2000
4	c3	1000
4	c1	1000
2	c5	2000

Stud-id + course-id ⟹ candidate key

course fee ⟹ non-prime attribute.

non-prime attribute is dependent on ~~non-pr~~ candidate key.

so This table conver 2NF, we need split this table in two table.

Stud-id	course-id

course-id	course fee

R(ABCDEF)

NOT in table

L.H.S proper subset of CK and

R.H.S non-prime attribute

FD {proper subset of CK

C → F → pd

E → A → Non prime attribute → pd

EC → D ✗

A → B

subset not proper subset

EC = FADB

$EC^+ = \{ECFADB\}$

(i) CK = {EC}

(ii) prime attributes {E, C}

(iii) NON prime {A, B, D, F}

3NF ⟺ partial ← Dependency → full → transitive

Transitive dependency:- If the value of a non-primary attribute can be defined using another non-primary attribute then it is called transitive dependency

Rules

- Table must be in 2NF
- Transitive functional dependency of non-pri attribute on any super key should be removed.

Employee table

emp_id	emp_name	emp_zip	emp_state	emp_city
101	A	1200	UK	nodia
102	B	2100	USA	chicago
103	C	8400	UP	Bhopal
104	D	9700	US	Norwich

candidate key = Emp_Id

~~Non~~ prime attribute = Emp-id

Non prime attribue = anothers key

Here emp_state and emp_city dependent on emp_zip.

and emp_zip dependent on emp_id

empid	emp name	emp_zip

emp-zip	emp-state	emp_city

Ex2

R(ABCD)

FD: $AB \rightarrow CD$, $D \rightarrow A$

R.S

CK: $AB^+ = ABCD$ $\Rightarrow$ CK: {AB, DB}

$DB^+ = DBAC$

prime Attributes = {A, B, D}

Non prime Attribute {C}

valid for 3NF

For each FD :

{ LHS must be a ck or SK OR
RHS is a prime attribute }

Here (AB) → CD   D → (A)

(AB → ck)   (A → prime)

so this table is in 3NF

BCNF

BCNF (α)

Non prime attribute ⟷ prime attribute

(Non prime attribute → super key convert)

- ~~must have s~~
- Table must be in 3NF
- A table is in BCNF if every functional dependency x → y , x is the super key of the table

• for BCNF, the table should be in 3NF and for every FD, LHS is super key

$x \rightarrow y$ (x must be super key)

Ex

$R(A, B, C)$

FD: $\{A \rightarrow B,\ B \rightarrow C,\ C \rightarrow A\}$ → it is 3NF

(A: SK, B: SK, C: SK)

super key:

$ABC^+ = \{ABC\}$

$AC^+ = \{ABC\}$

$A^+ = \{ABC\}$

$C^+ = \{ABC\}$

$B^+ = \{ABC\}$

CK = $\{A, C, B\}$

BCNF: ✓

valid for ...

Lec 24 Jennys

Example

Normalize a relation from 3NF to BCNF

R(ABCDEFGH)

FD: {
A → BD
B → C
E → FG
AE → H
}

Find CK

$ABCDEFGH^+ = \{ABCDEFGH$

SK ← $AE^+ = \{AEHBDCFG\}$

proper sub set

$A^+ = \{BDCA\}$

$E^+ = \{EFG\}$

CK

∴ prime attribute = {A, E}

non prime Attribute {BCDFGH}

Find PD partial dependency

LHS proper subset of CK (and)

RHS Non prime attribute

- ✓ $A \to BD \implies PD$
- $B \to C \implies \times$
- ✓ $E \to FG \implies PD$
- $AE \to H \implies \times$ (subset)

} so this table now 1NF

Now solve the Pd →

$A^+ = ABCD$

$R_1 = \{A, B, C, D\}$

$A^+ = ABDC$

$A \to BDC$

$B^+ = BC$

$B \to C$

~~$C^+ = C$~~ trivial

~~$D^+ = D$~~

$E^+ = EFG$

$R_2 = \{E, F, G\}$

$E^+ = EFG$

$E \to FG$

~~$F^+ = F$~~ $\times$

~~$G^+ = G$~~ $\times$

~~$R_3 = \{H\}$~~

$R_3 = \{H, A, E\}$

~~$H^+ = H$~~ $\times$

$A^+ = ABDC$ $\times$

~~$E^+ = EFG$~~ $\times$

$AE^+ = AEBDCFGH$

$AE \to H$

$AH^+ = AHBDC$ $\times$

FD_1 : { $A \to BDC$ (CK)
$B \to C \to$ Td
}

CK: A

FD_2 : {
$E \to FG$ (Superkey)

CK: E

BCNF

FD_3 : {
$AE \to H$ (SK)}

CK: AE

BCNF

$B^+ = BC$

$R_{11} = \{B, C\}$

$B^+ = BC$

$B \to C$ BCNF

$C^+ = C$ ✗

$R_{12} = \{A, D, B\}$

$A^+ = ABDC$ ✗

$A \to BD$ BCNF

$B^+ = BC$ ✗

$D^+ = D$ ✗

R(ABCDEFGH)

FD: { $A \to BD$, $B \to C$, $E \to FG$, $AE \to H$ }

$R_1(A, B, C, D)$
FD_1 : { $A \to BDC$, $B \to C$ }

$R_2(E, F, G)$
FD_2 : { $E \to FG$ }
BCNF

$R_3(AEH)$
FD_L : { $AE \to H$ }
BCNF

$R_{11}(B, C)$
FD_{11} : { $B \to C$ }
BCNF

$R_{12}(ADB)$
FD_{12} : { $A \to BD$ }
BCNF

'19-20 6(a)

R(a, b, c, d, e)

P(f, g, h, i)

a → b
b → c
b → e
b → d
f → g

P
f → h
h → i

accessionno = a
isbn = b
title = c
author = d
publisher = e
user id = f
name = g
dept id = h
dept name = i
R = books
P = users

1st NF: give schema is already 1NF because all attributes contain atomic values.

2NF: To achive 2NF there should bo partial dependency in FD.

in R(a,b,c,d,e) find ck

$ab\cancel{c}\cancel{d}\cancel{e}^+ = \{abcde\}$

so candidate key is A

prime attribute = {A}

Non prime attribute {B, C, D, E}

and, in P(F, G, H, I) fin ck

$FG\cancel{H}\cancel{I}^+ = \{FGHI\}$

ck = F

PA = {F}

NPA = {GHI}

find pd in both tables

A → B → ~~Bd~~	F → G
B → C	F → H
B → E	H → I
B → D	
no partial dependencies	No partial dependencies

So all relation have 2NF

3NF

To achive 3NF all transitive dependencies should be removed that means non prime attributes can not determine non prime attribute.

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

$A \rightarrow B$ → only valid

$B \rightarrow C$
$B \rightarrow E$
$B \rightarrow D$
} → TD

$R_1(A, B)$

$A^+ = AB$

$A \rightarrow B$

BCNF

$B^+ = BCDEK$

$FD_{R_1}: \{A \rightarrow B\}$

BCNF valid

$R_2(B, C, D, E)$

$B^+ = BCDE$

$B \rightarrow CDE$

$C^+ \Rightarrow C\alpha$

$D^+ = D\alpha$

$E = E\alpha$

$FD_{R_2}: \{B \rightarrow CDE\}$

BCNF

$P(F, G, H)$

$F \rightarrow G$

$F \rightarrow H$

$H \rightarrow I$ → TD

$P_1(H, I)$

$H^+ = HI$

$H \rightarrow I$

$I^+ = I\alpha$

$FD_{P_1}: \{H \rightarrow I\}$

BCNF

$P_2(F, G, H)$

$F^+ = FGHI$

$F \rightarrow GH$

$G = G\alpha$

$H = HI\alpha$

$FD_{P_2}: \{F \rightarrow GH\}$

BCNF

Transaction

A Transaction is a logical unit of work. It is the set of operations (read, write) to perform unit to work.

Operations of Transaction

① Read (x): A read operation is used to read the value of x from the DB and store in the buffer in the main memory for further actions.

② write (x): A write operation is used to write the value of the db from the buffer in the main memory

To withdraw money from Id 01 first read() operation then write() operation.

Balance

Id	name	Balance
01	x	500
02	y	1000

COMMIT is a transaction control language that is used to permanently save the changes don in the transaction in db.

RollBack: is a transaction control language that is used undo the transaction that have not been saved in the db

employee

Id	name	salary
1	Rahim	20000
2	Karim	30000

```
start transaction;
 update employee
 set salary = 50000
 where id = 3;

COMMIT; / ROllBACK
```

ROLLBACK → Return before state

COMMIT → successfully parmanently save in table

Save point: - This is used to set a point within a transaction to which you can later rollback if needed.

```
SAVEPOINT savepoint_name;
ROLLBACK TO savepoint-name;
```

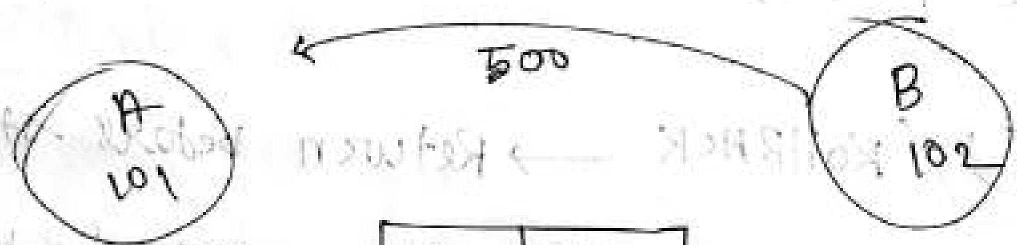
ACID

→ ~~Autom~~ Atomicity: The entire transaction takes place at once or doesn't happen at all

→ consistency: The database must be consistent before and after the transaction

→ Isolation: multiple transaction occurs independently without interference

→ Durability: The changes of successfully transaction occurs even if the system failure occurs. (commit)



1000+500 =1500

id	balance
101	1000
102	2000

2000 − 500 =1500

Transaction

before

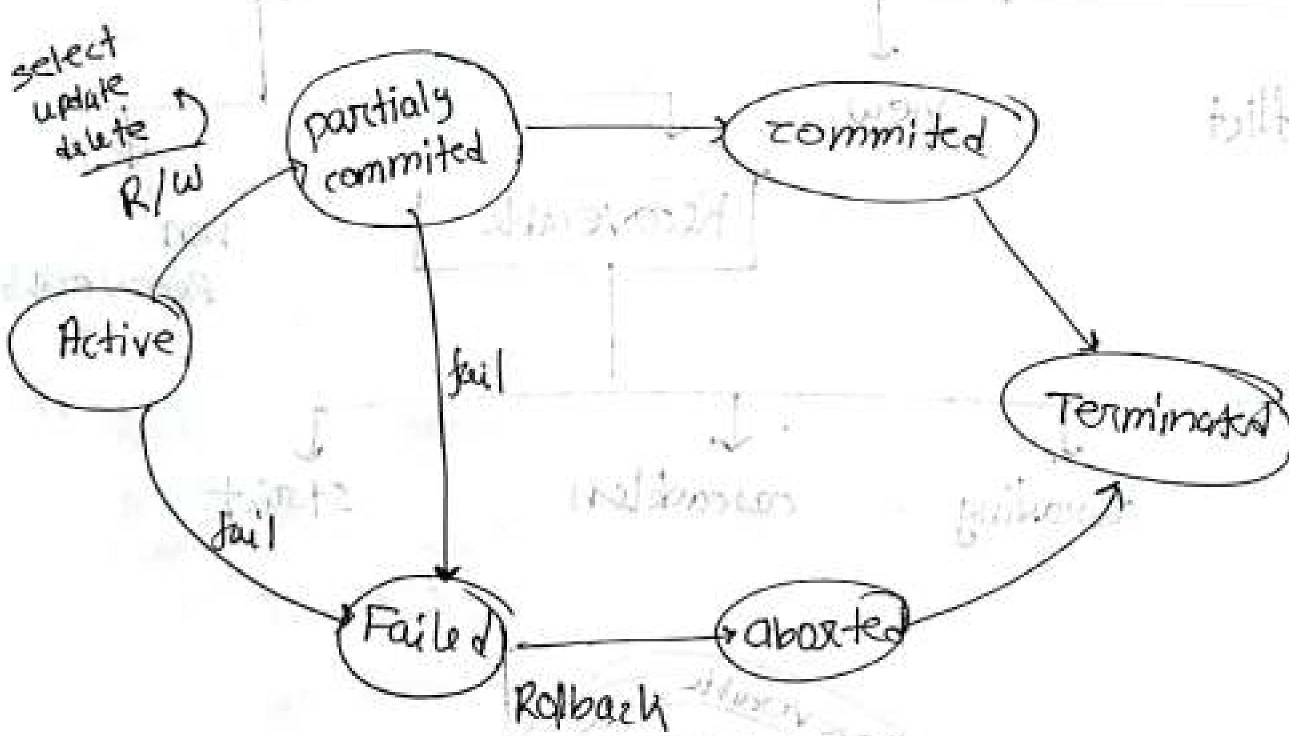
101 → 1000
102 → 2000
} = 3000

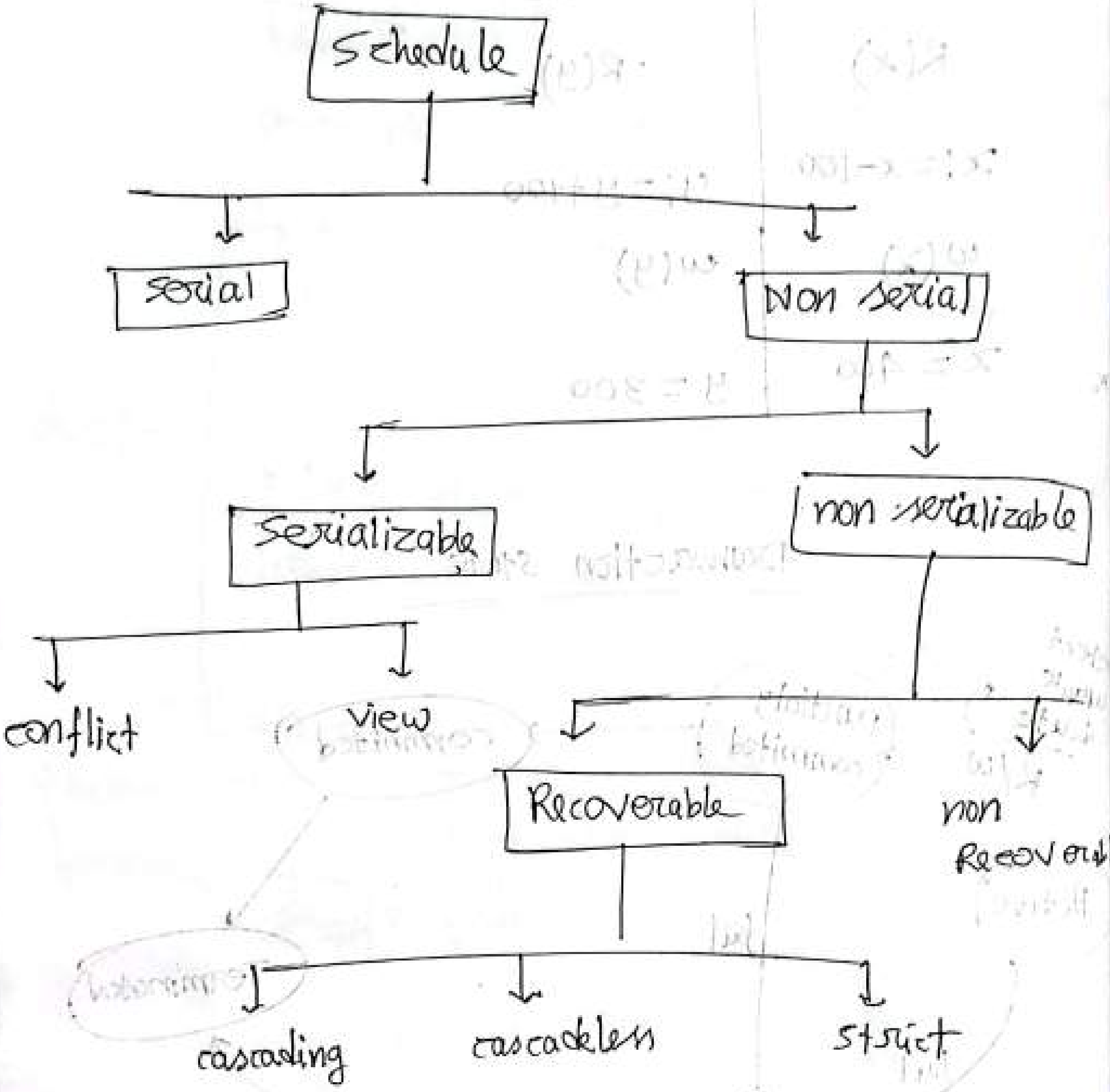
after

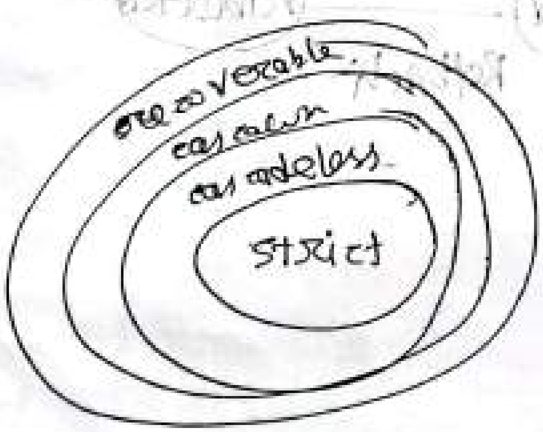
101 → 1500
102 → 1500
} = 3000

	T1	T2	
before $x = 500$			$y = 200$
	$R(x)$	$R(y)$	
	$x := x - 100$	$y := y + 100$	
	$w(x)$	$w(y)$	
After	$x = 400$	$y = 300$	

Transaction state




Schedule
serial
Non serial
Serializable
non serializable
conflict
view
Recoverable
non Recoverable
cascading
cascadeless
strict


strict

multi user support: concurrency

performance and speed: parallel

Advantages of concurrency

- decrease waiting time
- decrease responce time
- Resource utilization
- Increase efficiency

problems with concurrency

(parallel transaction)

Lost update problems (w – w conflict)

T_1: R(A), A = A+200, w(A)

A=1000

T_2: R(A), A = A−700, w(A)

	T_1		T_2	
1000	R(A)	t_1		
1200	A=A+200	t_2		
		t_3	R(A)	1000
		t_4	A = A − 700	300
1200	w(A)	t_5		
		t_6	w(A)	300

blind write

buffer

② Unrepeatable read problem

R-W conflict

T1	T2
1000 R(A)	
	R(A) 1000
	A = A − 200 800
	W(A) 800
800 R(A)	

Same dB: A = 1000

800 → problem

T1 depends on T2

③ Dirty read problem

W-R conflict

T1	T2
1000 ← R(A)	
← A = A+500	
1500 ← W(A)	
	R(A) → 1500
	A = A + 500
	W(A) 2000
R(B)	
failure	

A = 1000, 1500, 2000

RollBack: A = 1000 but A = 2000 present

Serial schedule

T_1	T_2
R(A)	
A:=A-500	
W(A)	
commit	
	R(A)
	A:=A+1000
	W(A)
	commit

Non serial schedule.

T_1	T_2
R(A)	
A:=A-500	
W(A)	
	R(A)
	A:=A+500
R(A)	
W(A)	

Recoverable

- → cascading
- → cascadeless
- → Strict

If some transaction T_j is reading or written be some other transaction T_i, then the commit of T_j must occure after the T_i commit.

② means if T_2 is dependent of T_1 then first commit of T_1 then T_2 commit

T_1	T_2	A = 1000
1000 R(A)		
R = A+500		
1500 W(A)		
	R(A) 1500	
	R = A + 500	
	W(A) 2000	
commit		
	commit	

cascading schedule

T_1	T_2	T_3
R(A) (source)		
A := A+500		
W(A)	(depend on T_1)	
	R(A)	
	A := A+500	
(Rollback)	W(A)	(depend on T_2)
R(B) (failure)		R(A)
commit	(Rollback)	A := A+500
	commit	W(A)
		(Rollback)
		commit

When failure in one T and this leads other T, rollback the source T also dependent T.

② cascadeless schedule

T_1	T_2
R(A)	
W(A)	
	W(A)
commit	
	R(A)
	W(A)
	commit

When a transaction is not allowed to read data until the last transaction which has written is commited

meam: if a T_1 read another T written data ensure that this written data is commited?

Here T_2 read updated tata of T_1 if only T_1 is commited

③ strict schedule

T_1	T_2
R(A)	
	R(A)
W(A)	
commit	
	W(A)
	R(A)
	commit

in strict schedule Read and write updated value of another T must ~~be~~ have commited.

Lec: 99 Farhan

Serializability schedule

conflict :- (i) can't loop/cycle

(ii) consistency of serial schedule

T1	T2	T3
→ R(A)		
	→ R(B) 20	
		→ R(B) 20
	→ W(B) 30	
→ W(A) 20		
		→ W(A) 30
	30 → R(A)	
	→ W(A) 40	

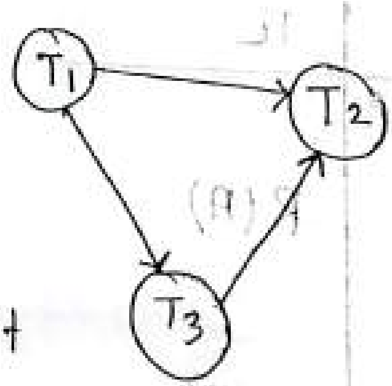
W(A) = 40
W(B) = 30

R — W
W — R
W — W

consistency check (valid)

A = 10, B = 20 { W(A) = A+10, W(B) = B+10 }

create a precedency graph

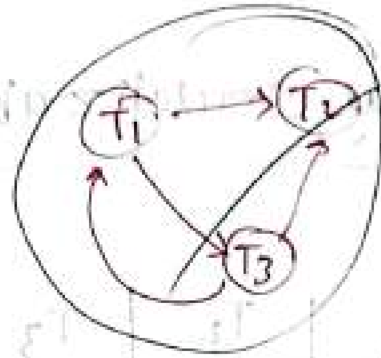


in this graph have not any loop or cycle so it is conflict schedule and it convert serial schedule.

$T_1 \rightarrow T_3 \rightarrow T_2$

Serial schedule

T1	T3	T2
R(A)		
W(A)		
	R(B)	
	W(A)	
		R(B)
		W(B)
		R(A)
		W(A)



non-conflict

- R(A)=40
- R(B)

W(A)=40
W(B)=70

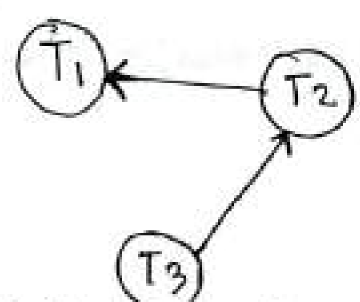
(19-20 7c)

S: $R_1(A)$; $R_2(A)$; $R_3(B)$; $W_1(A)$; $R_2(C)$; $R_2(B)$; $W_2(B)$; $W_1(C)$;

T1	T2	T3
→ R(A)		
	→ R(A)	
		→ R(B)
→ W(A)		
	→ R(C)	
	→ R(B)	
	→ W(B)	
→ W(C)		

R-W
W-R
W-W

conflict serializability schedule

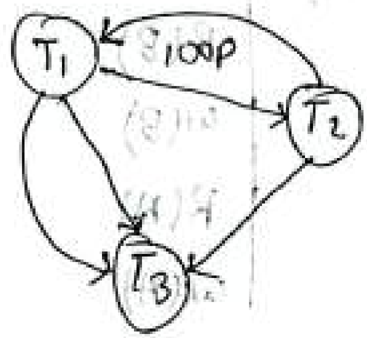


in the precedence graph has no cycle so it is conflict…

VIEW. serializability

R – W
W – R
W – W

T1	T2	T3
→ R(A)		
	→ W(A)	
→ W(A)		
		W(A)



difficult

18-19
85

① Transaction 1 (Transfer 50 from A to B)

START TRANSACTION;

UPDATE account set balance = balance-50 where account_name = 'A';

UPDATE account SET balance = balance+50 where account-name = 'B';

COMMIT;

② Transaction 2 (Transfer 10% of balance A to B)

START TRANSACTION;

UPDATE account SET balance = balance - (balance * 0.10) where account-name = 'A';

UPDATE account SET balance = balance + (balance * .10) Where account-name = 'B';

COMMIT;

18-19
4

(a) —

(b) (i) ck is AD

(ii) $AD^+ = \{ADBC$

(c)

Book_seller(ISBN_NO, salsman, sold date, comission, discound-amound)

consider that,

Book-seller = R

ISBN-NO = A

salsman = B.

sold-date = C

comission = D

discount = E

F. Dependencies

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

$A \to C$

$A \to E$

$C \to E$

$B \to D$

Find CK

$ABCDE^+ = \{ABCDE\}$

$AB^+ = \{ABCED\}$ → (CK)

$A^+ = \{ACE\}$

$B^+ = \{BD\}$

CK = AB

PA = {A, B}

NPA = {C, D, E}

1NF: all attributes are atomic so this relation is 1NF

2NF: • Relation is in 1NF

• ~~There are no partial dependencies in these~~ Relation

~~so now this relation also 2NF~~

~~2NF:~~ check partial dependencies:—

L.H.S is proper subset of ck and

RH.S is non-prime Attribute.

so partial dependencies relations are

$A \rightarrow C$

$A \rightarrow E$

$B \rightarrow D$

$A^+ = ACE$

$R_1(ACE)$

$A^+ = ACE$

$A \to CE$ — 2NF

$C^+ = CE$

$C \to E \longrightarrow$ Td

$CK = A$

$B^+ = BD$

$R_2(BD)$

$B^+ = BD$

$B \to D$ BCNF

$C^+ = CE$

$R_{11}(CE)$

$C^+ = CE$

$C \to E$

BCNF

$R_{12}(AC)$

$A^+ = ACE$

$A \to C$

BCNF

~~$R_P(A$~~

$R_1(B,D)$

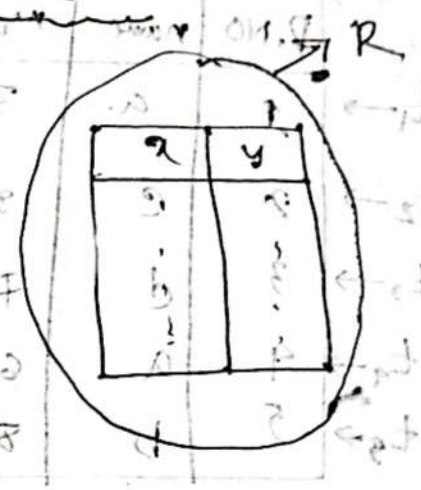
$R_2(C,E)$

$R_3(A,C)$

DBMS

Functional Dependency, keys

FD : $x \to y$ — x (determinent), y (dependent)



tapples = rows

** x এর মান যদি একরকম হয় y এর মানও same হতে হবে।

** If the tapple of x is same then y will also same values.

case 1

	x	y
$t_1 \to$	1	1
$t_2 \to$	2	1
$t_3 \to$	3	2
$t_4 \to$	4	5
$t_5 \to$	2	3

case 2

	x	y
$t_1 \to$	1	1
$t_2 \to$	2	1
$t_3 \to$	3	2
$t_4 \to$	2	1

▣ FD : $x \to y$

if $t_1.x = t_2.x$ then must be $t_1.y = t_2.y$ (case 2)

▣ FD : $x \to y$

if $t_1.x \neq t_2.x$ then you don't need to check the condition of y. y would be same or not same. (case 2)

But

▣ FD : $x \to y$,

if $t_1.x = t_2.x$ and $t_1.y \neq t_2.y$ then it is **not** the case of Functional dependency. case (1)

Example:

	R.NO	name	marks	Dept	course
$t_1 \to$	1	a	70	CSE	C_1
$t_2 \to$	2	c	80	EEE	C_2
$t_3 \to$	3	d	70	CSE	C_1
$t_4 \to$	4	a	68	ME	C_3
$t_5 \to$	5	b	81	EEE	C_2

is it FD?

R.no → name ✓ (x → y)

~~$t_1.x \neq t_2.x$~~

It is FD, 1 is true

Name → R.no ✗ (x → y)

$t_1.x = t_4.x$ but $t_1.y \neq t_4.y$ so: it is not FD

Dept → course ✓ (x → y)

$t_1.x = t_3.x$ and $t_1.y = t_3.y$

Again $t_2.x = t_5.x$

$t_2.y = t_5.y$

1. $t_1.x \neq t_2.x$ then no need to check y and it is a FD.

2. $t_1.x = t_2.x$ then must be $t_1.y = t_2.y$. then it will FD otherwise it is not FD.

3. Every time we just check the repeted value to determine FD. That's make our calculat easier.

R.no, Name ⟶ Marks ✓

x y

R.no.	name	mark
1	a	60 ←
2	b	70
3	d	80
4	e	90
1	a	61 ←

x y

R.no, name ⟶ marks

In this case not FD.

~~কোনো একটা condition এ FD এর কোনো~~

কোনো একটা case না মিললেই সেটা FD হবে না। অন্যগুলো check করা লাগবে না।

***** methods/properties.**

Reflecxivity: If y is a subset of x then $x \to y$ (trivial)

Augmentation: If $x \to y$ then $xz \to yz$

** Transitive: If $x \to y$ and $y \to z$ then $x \to z$

Union: If $x \to y$ and $x \to z$ then $x \to yz$

Composition: If $x \to y$ and $z \to w$ the $xz \to yw$

Decomposition: If $x \to yz$ then $x \to y$ and $x \to z$

Pseudutrasitive: If $x \to y$ and $wy \to z$ the $wx \to z$

Trivial FD: (~~Reflexivity~~)

if $x \to y$ and y is subset of x is called trivial FD and y is an attribute.

| This case always true. | Valid

L.H.S	R.H.S
id, name →	id
id, address →	address

$L.H.S \cap R.H.S \neq \phi$.

Non ~~trivial~~ trivial:

if $x \to y$ and y is not the subset of x is called non-trivial FD.

L.H.S	Right
id →	name
id →	semester

$L.H.S \cap R.H.S = \phi$

Trivial FD: (No need to check the table it always valid)

$x \to y$ if $y \subseteq x$	id, name → name
$x \to x$ if $x \subseteq x$	id → id

* very less use case

Non trivial FD:

$x \to y$ and $x \cap y = \phi$ and $y \notin x$. ~~but it is semitrivial.~~

id → name

x, y are totally different

if $\underset{x}{\underline{id, name}} \rightarrow \overset{\underline{name, Marks}}{y}$

x and y are different but here is an attribute common in L.H.S and R.H.S. This time it's call Semi trivial.

~~Armstrong's Axioms/In~~

Armstrong's Axioms / Inference rules:

1. Reflexivity: $x \rightarrow x$, $x \rightarrow y$ and $y \subseteq x$ (trivial)

2. Transivity: if ($x \rightarrow y$ and $y \rightarrow z$) then $x \rightarrow z$

Name → Dept
Dept → Faculty

Then name → Faculty.

3. Agmentention: if $x \rightarrow y$ then $xA \rightarrow yA$

id, city → name, city

4. Union: if $x \rightarrow y$ (id → name) and $x \rightarrow z$ (id → city) then $x \rightarrow yz$ (id → name, city)

5. Decomposition/splitting: if $x \rightarrow yz$ (id → name, city) then $x \rightarrow y$ (id → name) and $x \rightarrow z$ (id → city)

we cant split L.H.S.

But $xy \rightarrow z$ then $x \rightarrow z$ and $y \rightarrow z$ thats not possible.

Psedo transitive: If ($x \to y$ and $yz \to A$) then $xz \to A$.

Exp: id → name, name, city → marks then id, city → marks.

Composition: if $x \to y$ and $a \to b$ then $xa \to yb$

Attribute closure / closure set

** if you find all the closure set of attributes then it is easy to find candidate / super etc keys.

** If you find the candidate keys then you can easily do 2nd, 3rd, BCNF normalization table → R (A, B, C, D, E) → attributes

FD (A → B, B → C, C → D, D → E)

A → C
A → D
A → E } (transitivity)

A → ABCDE } (union)

A → B and B → C
So A → C (transitivity)
Also, A → A (Reflexivity)

B → C
B → D
B → E } transitivity

B → BCDE (union)

C → D
C → E (transitivity)
C → CDE (union)

D → E
D → DE

E → E

Closure: (X = set of attributes)

X^+ → Contains set of attributes determined by X

$A^+ = \{A, B, C, D, E\}$ → SK ✓

$AD^+ = \{A, D, B, C, E\}$ → SK ✓

$B^+ = \{B, C, D, E\}$ ✗

$CD^+ = \{C, D, E\}$ ✗

** super key: set of attributes whose closure contains all the attributes of given relation on table.

** Here A^+ is a super key, so AB^+, AC^+, AD^+, AE^+, ABC^+, $ABCD^+$ etc are also super key according to Augmentation rules.

So how many super key are there?

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

A → SK

B, C, D, E → super key বাদে বাকি যতগুলো attribute আছে তাদেরকে 2^n এর আকারে লিখলে যা হয় তাই হলো মোট super keys এর সংখ্যা। $2^4 = 16$

Candidate key:

Minimal super key, meaning no attribute can be removed without losing the ability to uniquely identify tuples

* A super key whose proper subset not to be a super key.

Exam: $A^+ = \{A, B, C, D, E\} \rightarrow SK$

$AD^+ = \{A, B, C, D, E\} \rightarrow SK$

Here proper subset of AD is {{A}, {D}} and {A}, is also a super key. So AD cannot be a candidate key.

(So only {A} is a candidate key.)

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

FD (A → B, D → E)

So closure set:

$A^+ = \{A, B\}$ ✗

$BC^+ = \{B, C\}$ (Reflexivity) ✗

$ABCDE^+ = \{A, B, C, D, E\}$ (Super key)

$ABDE^+ = \{A, B, D, E\}$ ✗

$ACDE^+ \rightarrow \{A, C, D, E, B\}$ SK ✓

$ACD^+ \rightarrow \{A, C, D, B, E\}$ SK ✓

$AD^+ \rightarrow \{A, D, B, E\}$ ✗

$CD^+ \rightarrow \{C, D, E\}$

$AC^+ \rightarrow \{A, C, B\}$

candidate key:

$ABCDE^+$ এর proper subset এ ACDE, ACD etc চলে আসে যারা super keys. So $ABCDE^+$ candidate key হবে না। As same ACDE.

But ACD^+ এর proper subset:

A	AD
C	CD
D	AC

যার কোনোটিই SK নয়

So, ACD^+ is a candidate key.

how many candidate keys are there?

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

FD → {A → B, D → E}

1. $ABCDE^+ = \{A, B, C, D, E\}$ SK
2. $ACDE^+ = \{A, B, C, D, E\}$ SK

$ACD^+ = \{A, B, C, D, E\}$ SK

$AC^+ = \{\}$
$CD^+ = \{$
$AD^+ = \{$
$A^+ = \{\}$
$C^+ = \{\}$
$D^+ = \{\}$

none of SK

সহজে candidate key বের করব:

① R( ) Table এর অ্যাট্রিবিউট নিয়ে একটা closure set বের করব যেটা অবশ্যই SK হবে।

② FD অনুযায়ী যেসব attributes বাদ দিব যাদেরকে অন্য attributes দিয়ে determine করা যায়।

③ অবশেষে এমন একটি closure অবস্থায় আসবে যার কোনো attribute আর বাদ দেওয়া যায় না।

④ তখন check করব অবশিষ্ট closure এর proper subset এর কোনো SK থাকবে না, না হলে এটাই candidate key হবে।

So, candidate key is ACD^+ and there are no more ck.

Prime Attributes: (part of candidate Keys) FD(A→B, D→E)

candidate key is [ACD+]

So prime attributes {A, C, D}

* Now check Are prime attri. available on the R.H.S of the FD? If ans. is NO. then there is no more candidate keys in the Relation.

So only candidate key is ACD.

Exmp: 2 R(A, B, C, D)

FD → {A→B (i), B→C (ii), C→A (iii)}

step 1: $ABCD^+ = \{A, B, C, D\}$ → SK

step 2:

(i) $ACD^+ = \{A, C, D, B\}$ → SK

(ii) $AD^+ = \{A, D, B, C\}$ → SK

proper subset of AD:
$A^+ = \{A, B, C\}$
$D^+ = \{D\}$
} not SK

Now we need to find any other ck is exist or not.

* So candidate key is AD^+.

Prime Attributes are {A, D}, [check this] [এখানে দেখবো এ Attri এর কোনো FD এতে R.H.S আছে কি না] Yes, C → A

A R.H.S এ আছে, so we can write, AD

as CD because C can determine A,

A D
↓
C D → SK

SK ✗ → $C^+ = \{C, A, B\}$ } Now check is it a ck?

SK ✗ → $D^+ = \{D\}$

proper subset এ কোনো SK নেই।

So we can say CD is also a CK.

So the final prime Attributes are {A, D, C}.

আবার, CT করে দেখবো,

Yes B → C, C is on the R.H.S. So we can write,

AD → CK
↓
SK → CD → CK

SK → BD → Now check if it a ck?

$B^+ = \{B, C, A\}$ } none of SK
$D^+ = \{D\}$

So. BD is a CK

So, the Prime attn. = {A, D, C, B}

again CT করবো। Yes $A \to B$ B is on the R.H.S. So we can write,

AD → CK
↓
SK → CD → ck
↓
SK → BD → CK
↓
SK → AD → CK } we already check this.

So, All the ck are = {AD, CD, BD}

and prime attributes = {A, B, C, D}

There is no non-prime attributes.

Example:

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

$FD = \{AB \to C, C \to DE, E \to F, D \to A, C \to B\}$

$ABCDEF^+ = \{A, B, C, D, E, F\} \to SK$

$ABDEF^+ = \{A, B, C, D, E, F\} \to SK$

$ABF^+ = \{A, B, F, C, D, E\} \to SK$

$AB^+ = \{A, B, C, D, E, F\} \to SK$

$A^+ = \{A\}$
$B^+ = \{B\}$] none of SK

So AB is a CK.

Prime Attributes $\{A, B\}$

AB → CK
↓
SK → DB

$D^+ = \{D, A\}$

$B^+ = \{B\}$

So DB is a CK

prime attri $\{A, B, D\}$

AB → AC → SK
↓
DD

$A^+ = \{A\}$ SK

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

So, AC not a CK.

But C is a CK.

$P.A = \{A, B, D, C\}$

AB — AC
↓
DB
↓
not CK → CB → SK

$C^+ = \{ \quad \} SK$

$B^+ = \{ \quad \}$

CK = { AB, BD, C }

PA = { A, B, C, D }

non-PA = { E, F }

Canonical Cover

A canonical cover (or minimal cover) of a set of FD is simplified version of that set where

1. Redundent dependencies are removed
2. " attributes within dependencies are removed
3. each dependency is a single attribute dependency on the right side.

F' if F' don't have

① extraneous attribute / redundant att

(ii) Redundent FD.

To make this.

Step ① Spliting rule / Decomposition

So that in every R.H.S has single attribute

Step ② ** You can't split L.H.S **

Example: A → BC So A → B and A → C

Step: (ii) Remove redundent attribute.

Example:

$F_1 = \{ AB \to C, \underline{A \to C} \}$

Here we can define C by A alone
So no need to define C by AB.

eg same $\{ AB \to C, \underline{A \to B} \}$ | we can remove B from AB

$F_2 = \{ AB \to C, \underline{B \to C} \}$

no need to AB.

(we can remove A from AB)

(iii) Remove redundent/duplicate FD.

Example:

F :- { AB → C, C → AB, B → C, ABC → AC, A → C, AC → B }

Step 1: :- { ~~AB → C~~, C → A, C → B, B → C, ~~ABC → A~~, ~~ABC → C~~, A → C, ~~AC → B~~ }

Step 2: :- { ~~B → C~~, C → A, ~~C → B~~, B → C, ~~A → C~~, ~~A → B~~ }

(C → B: we can find B indirectly by A → B)

Step 3: :- { C → A, B → C }, A → B

$C^+ = C, B$ — we cannot get A here so we cannot remove C → A

$C^+ = C, A, B$

$B^+ =$

$A^+ = A, B, C$

$A^+ = A, C$

** Canonical Cover এর answer কখনো same নাও হতে পারে। Person to person vary করবে। **

Semester Q. Solve

Topic Functional Dependency: (20-21 final exam) {19-20 Q. 1(b) / 18-19 Q. 2(d)}

Q: 2(a) / Let R = (A, B, C, D). If AB and BD can uniquely identify a tuple in a relation r(R) separately then how many super keys, ck and PK are there?

Answer: candidate key:

We are given that AB and BD can each uniquely identify a tuple in R without any redundancy.

Since AB and BD uniquely identify tuples and contain the minimal attributes with uniqueness, So they are the candidate key of that relation.

So the number of candidate keys are two. these are = {AB, BD}.

Super key: Super keys are sets of attributes that can uniquely identify a tuples.

They include ck and any superset of those ck.

We already found that candidate keys are AB and BD.

So ~~the~~ according to Augmentation Rules we can say all the combination of attributes with cks are also super keys.

Any combination with AB : (ABC, ABD, ABCD)

Any " " BD : (BDA, BDC, BDAC)

So the superkeys are,

{AB, BD, ABC, ABD, ~~ABCD~~, BDC, ABCD}

number of super keys are 6.

Primary key: A primary is a key that can uniquely identify any tuples in a Relation. In a relation there will be only one PK.

So PK is = {AB or BD}   num. of PK is 1.

6(a)

~~Assumptions~~

~~The~~ Given that,

employee(emp-id, emp-name, emp-phone, dept-name,
dept-phone, dept-mgr id, skill-id, skill-name
skill-date, skill-level)

Assumptions:

1. emp-id uniquely identifies each employee.
2. dept-name identifies dept-mgr id which is unique to each department.
3. skill-id uniquely identifies each skill-name
4. combination of emp-id and skill-id can uniquely identifies ~~all~~ ~~of~~ emp-name, ~~emp-phone~~ skill-level, skill-date, skill-name.
5. dept-mgr id uniquely identifies dept-name, dept-phone.

So the Functional Dependencies are.

FD 1 : emp-id → emp-name, emp-phone, dept-name

FD 2 : dept-name → ~~dept-pho~~ dept-mgrid, dept-phone

FD 3 : dept-mgrid → dept-name, dept-phone

FD 4 : skill_id → skill_name

FD 5 : (emp-id, skill-id) → emp-name, ~~emp-phone, dept-name~~, skill-name, skill-date, skill-level.

These are all the dependencies.

6 (a) (i) ~~Functional Dependency set~~:

Assumption:

① An employee can have multiple skills, but each skill is unique to an employee.

② A skill can be possessed by multiple employee

③ A department has a unique manager.

Functional dependency set:

1. Employee Information: ① emp-id → emp-name, emp-phone, dept-name

~~emp-id → dept-name~~

2. Department Information:

dept-name → dept-phone, dept-mgr id

3. Skill information:

skill-id → skill-name

4. Employee-skill relationship:

(emp-id, skill-id) → skill-date, skill-level.

(ii) To normalize the database into BCNF we can decompose it into the following relations.

1. Employee (emp-id, emp-name, emp-phone, dept-name)

primary key: emp-id.

2. Department (dept-name, dept-phone, dept-mgrid)

Primary key: dept-name

3. Skill (skill-id, skill-name)

Primary key: skill-id

4. Employee skill (emp-id, skill-id, skill date, skill-level)

Primary key: (emp-id, skill-id)

This decomposition ensure that each relation is in BCNF. The first three relations are already in BCNF as they are atomic and have simple primary key. The 4th relation, (Employee_skill) is also in BCNF because the primary key (emp-id, skill_id) uniquely determines all other attributes.

This normalized design remove all of the redundancy and ensure data integrity.

Q. id → name, desig, email.

name, desig → email, salary

name → email

~~email → id~~ id

~~To compute canonical cover we should follow these steps.~~

(B) Given the functional dependency (FDs),

1. id → name, designation, email
2. name, designation → salary, email
3. name → email
4. email → id

* Now consider id as 'A', name as "B", designation as 'C', email as 'D', salary as 'E' to easier the calculation.

So, 1. A → B, C, D
2. B, C → E, D
3. B → D
4. D → A

Step 1: Decomposition

Decompose all the FDs so that every FDs has a single value on the R.H.S.

1. A → B
2. A → C
3. A → D
4. (B, C) → E
5. BC → D
6. B → D
7. D → A

Step 2: Remove Etraneous Attributes from

FD 4: BC → E

if we remove C ~~then can we~~

B alone can't determine E.

again if we remove B

C alone can't determine E

So BC → E remain as is.

FD 5: BC → D

if we remove C

B can determine D

So, we can remove C and the FD remains B → D

After the step 2 we get the FDs are,

1. A → B
2. A → C
3. A → D
4. BC → E
5. B → D
6. D → A

Step 3: Remove Redundant FDs

FD 3: if we remove A→D,

we can still get A→D from

D→A and {A→C, A→B}

So A→D is a redundant FD.

The final FDs are →

1. A→B = id→name
2. A→C = id→designation
3. BC→E = (name, designation)→salary
4. B→D = name→email
5. D→A = email→id

These are the final canonical cover.

F:- (id→name, id→designation, (name, designation)→salary, name→email, email→id)

7(a)

19-20

(b)

Given that,

Relation schema R = {A, B, C, D, E}

FD: A → BC, CD → E, B → D, E → A

Now, the closure set of following are

$A^+ = \{ABCDE\}$

$(AB)^+ = \{A, B, C, D, E\}$

$(BC)^+ = \{B, C, D, E, A\}$

$(ABC)^+ = \{A, B, C, D, E\}$.

All of those are super keys. ~~and the minimal of them is A, so A~~

(19-20)

6 (b)

(i) AB → C:

AB→C that means A,B determines C and ~~every time~~ if row with the same value of A and B then have the same value for C.

In rows 1 and 3 we have identical ~~values for C but different~~ values for A=1 and B=2 but different for C (in row 1: C=3 and in row 3, C=4)

Result: As same value of AB doesn't match with C in rows 1 and 3, so AB→C doesn't holds in relation.

(ii) B→D:

That means B determines D and if rows ~~with~~ of B are same value then D should be same values.

In rows 1, and 3 we see B=2 and D=4 and in row 2 ~~B=~~ B=4 and D=4

Result: For every values of B the values of D remain same. So B→D holds in relation.

(iv) DE→A:

If the same value of D and E have the same value for A exist then it will be hold in relation.

In rows 1 and 2 identical values of D=4 and E=2 and both have A=1.

Result: DE→A holds because rows with identical values of D and E have a consistent value for A.

6(c)

~~Given that Function~~

Given Relation and Functional dependency:

R(A,B,C,D)

FDs : 1. AB → CD

2. BC → D

Step 1: Decompose R.H.S:

FDs:

1. AB → C
2. AB → D
3. BC → D

Step 2 Remove Redundant Attributes:

1. For AB → C:

(i) Removing A: $B^+ = \{B\}$ doesn't include C, so A is not redundant

(ii) Removing B: $A^+ = \{A\}$ doesn't include C, so B isn't redundant.

2. For AB→D:

(i) Removing A : $B^+ = \{B\}$ doesn't include D, so A is not redund.

(ii) Removing B : $A^+ = \{A\}$ doesn't include D, so B is not redun

3. For BC→D:

(i) Removing B: $C^+ = \{C\}$ ~~doesn't~~ include D, so B is not redundant

(ii) Removing C : $B^+ = \{B\}$ " " D, so C " " "

So, all the dependencies ~~are not re~~ has no redundant attributes.

Step 3: Remove Redundant FDs

Since BC→D is not implied by AB→C and AB→D

So it is not redundant.

Final Minimal cover for R(A, B, C, D) is:

1. AB→C
2. AB→D
3. BC→D.

18-19

4(a) Is it important to have a FD in each table? Why or why not?

Ans: No, it is not strictly necessary to have a functional dependency (FD) in each table. However, FDs are important for defining the relationships between attributes and ensuring data integrity. They help with normalization, reducing redundancy and eliminating update anomalies.

In case where no FDs exist, the table may lack meaningful structure and could lead to data anomalies.

(b) (i)

Given the dependencies are,

1. A → BC Relation, R(A, B, C, D, F)

2. B → E

3. CD → EF

closure set:

ABCDF+ = {A, B, C, D, F, E} → super key (SK)

ADF+ = {A, D, F, B, C, E} → sk [BC can determine by A]

Now the subset of ADF.

A+ = {A, B, C, E} → not a sk

D+ = {D} → not a sk

F+ = {F} → not a sk

AD+ = {A, D, B, C, E, F} → SK

~~AF+ = {A, F, B, C, E} → not a sk~~

~~DF+ = {~~

~~As AD is a super key, so the AD~~

As AD is a super key and the minimal of all super keys, so it is the candidate key.

(11) Prove that $AD \to F$ Holds in R.

To prove that $AD \to F$ holds, we can use the closure of AD to see if it includes F

closure of AD^+:

$(AD)^+ = \{A, D\}$

Using $A \to BC$, we get $AD^+ = \{A, D, B, C\}$

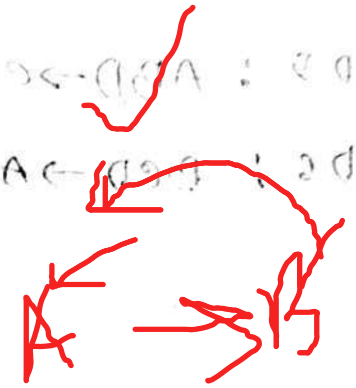
Using $B \to E$ we get $AD^+ = \{A, D, B, C, E\}$

Using $CD \to EF$ we get $AD^+ = \{A, D, B, C, E, F\}$

Since AD^+ includes F, the dependency $AD \to F$ holds in R.

6(d) List all the non-trivial FDs:

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_2	c_2	d_3
a_3	b_3	c_2	d_4



Non-trivial FDs:

Non-trivial FDs are those where dependent doesn't the part of of ~~dep~~ determinant.

Example: A → B , CD → E etc.

But A → A , CD → D are trivial FDs.

The non-trivial FDs are given below:

~~FD1: A → C~~

~~a1 maps c1 and a2 maps~~ c2 both times.

~~FD2: A → B~~

~~a1 maps b1 and~~

~~FD2: B → C~~

~~b1 maps~~

~~FD2: AB → C~~

~~a2 and b2 maps~~ c_2

~~both times.~~

FD 1: A → C

FD 2: AB → C

FD 3: AD → C

FD 7: D → B

FD 4: CD → A

FD 8: AD → B

FD 9: BC → A

FD 5: ABD → C

FD 10: CD → B

FD 6: BCD → A

FD 11: ACD → B

Utrozet : indexing & SQL

INDEX

Index:

Index is the data structured technique that helps to retrive data quickly from database.

Syntax:

Create indexing index_name
ON table_name (column 1, column 2, ...);

Type of indexing

- Ordered indices
  ↳ Base of sorted ordering values
- Hash indices
  ↳ base on values determine by function called hash function

Structure:

Search Key	Data Reference pointer

- Search Key ↳ Primary candidate key sorted order
- Data Reference pointer ↳ Pointer holding the address of the disk block.

S.K	D.R
1	1001
2	1004
3	1006
4	1008

indexing methods →
- Ordered
- Primary → Dense index / sparse index
- clustering
- Secondary

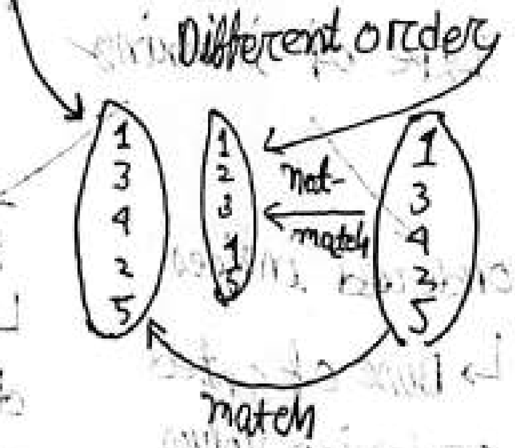
i) Ordered indices:

In an ordered index, index entries are stored on the search key value

Primary/clustering index

In search key of the index has some order as the sequential order of the file.

Secondary/Non-clustering index



ii) clustered indexing:

In this two or more columns are grouped together to uniquely identify the records.

↳ Records with similar characteristics are grouped together and indexes are created for this groups.

Ex.

Dept_id	ename
1	A
1	B
2	C
2	D
3	E
4	F

Dept_id	pointer
1	1000 → [1 A, 1 B]
2	1010 → [2 C, 2 D]
3	1020
4	1030

Primary index:

Data is stored according to the search key (Primary key to table)

↳ allowes sequential file organization.

↳ Primary key is used to create index.

↳ Efficient

Types:-

Dense Index

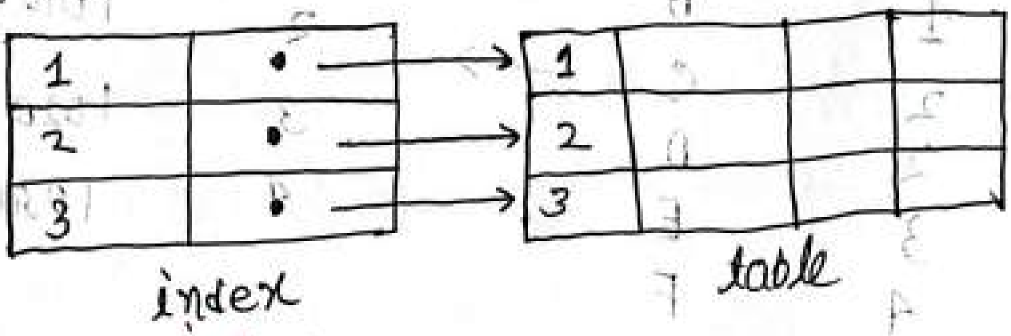
Index record for every search key value.

Sparse Index

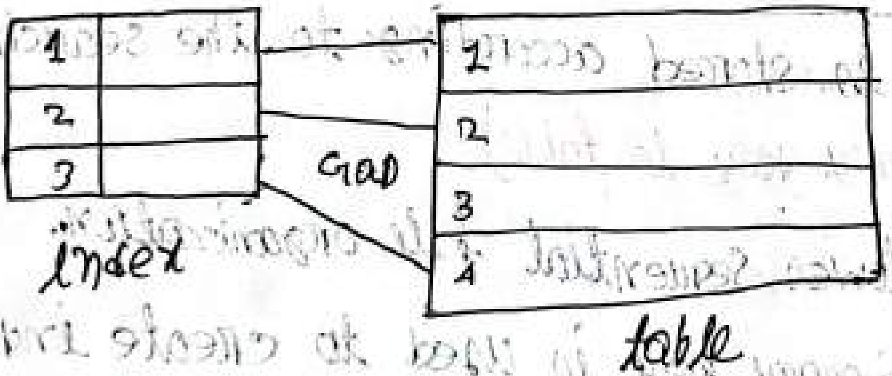
Index record for only few item.
Each item points to Block.

Dense index:

No. of records is same as no. of indexes
↳ in table



Sparse index:



Secondary index:

It is used to optimize query processing and access records in database with some information other than primary key.

Levels used:

First Level

Large range of no. is selected.

Second Level

Actual physical loc^n of data

eid	pointer
1	→
2	→
500	→
2000	→

280

eid	Pointer
200	
400	
600	

First level

200	→ 200
250	→ 260
300	
350	
400	
600	

Second level

Question 20-21

Topic: indexing

8.b) What is sparse indexing? How does multilevel indexing improve the efficiency of searching an Index file?

Solve:

Sparse Indexing:

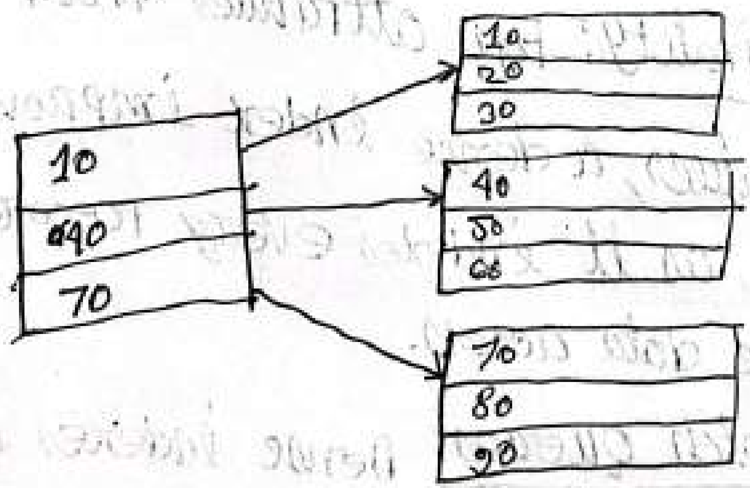
In sparse indexing, only certain records are indexed - usually the first record of each block. This allows a smaller index file and faster ~~se~~ search times within the index.

Once the nearest index entry is found, the ~~se~~ search continues within that block to locate the exact record.

~~Mutli~~

Multilevel indexing:

Multilevel indexing improves efficiency by creating a hierarchical structure of indexes, where the first level indexes point to second-level indexes and so on. This reduces the number of disk accesses needed to locate data, as the search progresses down the level of indexes rather than scanning a single large index. The hierarchical structure allows faster data retrieval especially for large databases.



c) When is it preferable to use a dense index rather than a sparse index? Explain your answer

Solve:

A dense index is preferable to a sparse index in the following scenarios:

1. Frequent searches: If the application performs many searches, a dense index allows for quicker lookups since it contains an entry for every search key, enabling direct access to records

2. High cardinality: For attributes with many unique values, a dense index improves retrieval efficiency, as it indexes every record, allowing for precise data access.

3. Exact Match queries: Dense indexes are optimal for ~~eax~~ exact match queries because they provide direct access to all records, enhancing performance in these case.

8d) Now make necessary modification to the index file after deletion of the record for the account no 'A-5' and then 'A-2'

Index file after delation of the record for the account no 'A-5':

Branch name	Pointer
Adabor	→ A-9
Dhanmondi	→ A-8
Mirpur	→ A-2
Motijheel	→ A-6

Index file after deletion of the record for the account no 'A-2'

Branch-name	Pointer
Adabor	→ A-9
Dhanmondi	→ A-8
Mirpur	→ A-4
Motijheel	→ A-6

Q-2019-20

8(a) Here's a brief differentiation for each type of indexing:

1. Primary indexing: An index base on a table's unique primary key; it organizes records in order and allows fast access.

2. Secondary indexing:

An index on non-primary, non-unique fields; allows multiple indexes per table for quicker lookups on specific columns.

3. Clustering indexing:

Build on non-unique columns with repeated values; groups similar records together, which is efficient for range searches.

b) 2021-21(c)

c) In general, it is not possible to have two primary indices on the same relation for different keys because the tuples in a relation would have to be stored in different order to have same values stored together. We could accom-plish this by storing the relation twice and duplicating all values, but for a centralized system, this is not efficient.

d) 1) Instance of Relation:

Course relation:

Course-name	room	instructor
Math 101	R_1	Prof. Smith
Physics 101	R_2	Prof. Johnson

enrollment Relation:

Courses_name	Student-name	grade
Math101	Alice	A
Math 101	Bob	B
Math 101	Carol	C
Physics 101	Dave	B
Physics 101	Eve	A
physics 101	Frank	C

Clustering structure:

Data is clustered based on courses-name storing each courses and its corresponding enrollment records together:

- cluster 1 (Math 101): (Math 101, R1, Prof. smith), (Math 101, Bob, B), (Math 101, Carol, C)
- cluster 2 (Physics 101): (physics 101, R_2, Prof. Johnson), (Physics101, Dave, B), (Physics101, Eve, A), (Physics101, Frank, C).

2020-21

a) Consider the database schema below: [10]

employee (ename, street, city)

emp_company (ename, cname, salary, jdate)

company (cname, city)

manager (ename, mname, shift)

Note: A manager is also an employee of a company.

Give SQL and RA expressions for the following queries:

→ R.A not needed this is out of syllabus.

(i) Find names, street addresses and cities of residence of all employees who work under manager Sabbir and who joined before January 01, 2019.

(ii) Find the names of the employees living in the same city where Rahim is residing.

(iii) Display the average salary of each company except Square Pharma.

(iv) Increase the salary of employees by 10% for the companies those are located in Barisal.

(v) Delete records from emp_company that contain employees living in Rajshahi.

b) SQL allows a foreign-key dependency to refer to the same relation, as in the following example: [2]

```
CREATE TABLE manager
    (employee-name  CHAR(20),
    manager-name  CHAR(20),
    PRIMARY KEY employee-name,
    FOREIGN  KEY  (manager-name)  REFERENCES  manager(employee-
    name) ON DELETE CASCADE);
```

Here, *employee-name* is a key to the table *manager*, meaning that each employee has at most one manager. The foreign-key clause requires that every manager also be an employee. Explain exactly what happens when a tuple in the relation *manager* is deleted.

(i) Find names, street addresses, and cities of residence of all employees who work under manager Sabbir and who joined before January 01, 2019.

SQL Query:

Sql Copy code

```sql
SELECT e.ename, e.street, e.city
FROM employee e
JOIN emp_company ec ON e.ename = ec.ename
JOIN manager m ON e.ename = m.ename
WHERE m.mname = 'Sabbir' AND ec.jdate < '2019-01-01';
```

Relational Algebra:

$$\pi_{ename,street,city}\left(\sigma_{mname='Sabbir' \wedge jdate<'2019-01-01'}(employee \bowtie emp\_company \bowtie manager)\right)$$

(ii) Find the names of the employees living in the same city where Rahim is residing.

SQL Query:

Sql

```sql
SELECT e1.ename
FROM employee e1
JOIN employee e2 ON e1.city = e2.city
WHERE e2.ename = 'Rahim';
```

Relational Algebra:

$$\pi_{e1.ename}\left(\sigma_{e1.city=e2.city \land e2.ename='Rahim' \land e1.ename \neq 'Rahim'}(employee \times employee)\right)$$

(iii) Display the average salary of each company except Square Pharma.

SQL:

Sql

```sql
SELECT cname, AVG(salary) AS average_salary
FROM emp_company
WHERE cname != 'Square Pharma'
GROUP BY cname;
```

Relational Algebra:

$$\gamma_{cname, AVG(salary)}\left(\sigma_{cname \neq 'Square Pharma'}(emp\_company)\right)$$

(iv) Increase the salary of employees by 10% for the companies that are located in Barisal.

SQL:

Sql

```sql
UPDATE emp_company
SET salary = salary * 1.1
WHERE cname IN (SELECT cname FROM company WHERE city = 'Barisal');
```

Relational Algebra:

Relational Algebra does not have an update operation, so it is not expressible in traditional RA.

(v) Delete records from `emp_company` that contain employees living in Rajshahi.

SQL:

Sql

```sql
DELETE FROM emp_company
WHERE ename IN (SELECT ename FROM employee WHERE city = 'Rajshahi');
```

b) When a tuple (row) in the manager table is deleted, ON Delete cascade triggers an automatic deletion of any rows where that tuple's employee_name is referenced as manager-name. This deletion process continues recursively, ensuring that any employees managed directly or indirectly by the deleted employee are also removed, preserving referential integrity within the table

Shortest:

When a row in the manager table is deleted, ON DELETE CASCADE automatically deletes all rows where that employee is listed as a manager, continuing recursively until no references remain.

5. a) Consider the database schema below: [8]

worker (wname, street, city)
works (work_id, wname, orgname, salary, jdate)
organization (orgname, city)
manages (wname, manager-name, shift)

Note: A manager is also an employee of an organization.
Give SQL expressions for the following queries:

(i) Find the names of all employees who work for "Google".
(ii) Find the names of all employees in this database who live in the same city as the company for which they work.
(iii) Find the names of all employees who live in the same city and on the same street as do their managers.
(iv) Give all managers in the database a 7.5% salary raise.
(v) Find the names of the employees living in the same city where Rahim is residing.
(vi) Find the company with the most employees
(vii) Create a view to show all the employees who earn more than average salary.
(viii) Find all the employees who work more than five years for "Facebook".

(i) Find the names of all employees who work for "Google".

Sql

```sql
SELECT wname
FROM works
WHERE orgname = 'Google';
```

(ii) Find the names of all employees who live in the same city as the company for which they work.

Sql

```sql
SELECT w.wname
FROM worker w
JOIN works ws ON w.wname = ws.wname
JOIN organization o ON ws.orgname = o.orgname
WHERE w.city = o.city;
```

(iii) Find the names of all employees who live in the same city and on the same street as their managers.

Sql

```sql
SELECT w.wname
FROM worker w
JOIN manages m ON w.wname = m.wname
JOIN worker wm ON m.manager-name = wm.wname
WHERE w.city = wm.city AND w.street = wm.street;
```

(iv) Give all managers in the database a 7.5% salary raise.

Sql

```sql
UPDATE works
SET salary = salary * 1.075
WHERE wname IN (SELECT wname FROM manages);
```

(v) Find the names of the employees living in the same city where Rahim is residing.

Sql Copy code

```sql
SELECT wname
FROM worker
WHERE city = (SELECT city FROM worker WHERE wname = 'Rahim');
```

(vi) Find the company with the most employees.

Sql Copy code

```sql
SELECT orgname
FROM works
GROUP BY orgname
ORDER BY COUNT(wname) DESC
LIMIT 1;
```

(vii) Create a view to show all the employees who earn more than the average salary.

Sql Copy code

```sql
CREATE VIEW AboveAverageSalary AS
SELECT wname, salary
FROM works
WHERE salary > (SELECT AVG(salary) FROM works);
```

(viii) Find all the employees who have worked for "Facebook" for more than five years.

Sql Copy code

```sql
SELECT wname
FROM works
WHERE orgname = 'Facebook' AND DATEDIFF(CURDATE(), jdate) > 5 * 365;
```

b) SQL allows a foreign-key dependency to refer to the same relation, as in the following example: [4]

```
CREATE TABLE manager(
employee-name  CHAR(20),
manager-name  CHAR(20),
PRIMARY KEY employee-name,
FOREIGN  KEY  (manager-name)  REFERENCES  manager(employee-
name) ON DELETE CASCADE);
```

Here, *employee-name* is a key to the table *manager*, meaning that each employee has at most one manager. The foreign-key clause requires that every manager also be an employee.

(i) Explain exactly what happens when a tuple in the relation *manager* is deleted.

(ii) What will happen, if ***RESTRICT*** is used instead of ***CASCADE***?

(i) When a tuple (row) in the `manager` table is deleted with `ON DELETE CASCADE`, any rows that reference the deleted `employee-name` as their `manager-name` will also be deleted automatically. This deletion will continue recursively, ensuring that all employees who directly or indirectly report to the deleted employee are also removed, preserving referential integrity.

(ii) If `RESTRICT` is used instead of `CASCADE`, deletion of a tuple is restricted if it has dependent rows (i.e., if other rows reference its `employee-name` as their `manager-name`). In this case, attempting to delete such a row will produce an error, preventing the deletion until all dependent rows are removed or updated to maintain referential integrity.

5. a) Consider the following table and give an expression in SQL for each of the following queries: [10]

employee(employee_name, street, city)
works(employee_name, company_name, salary)
company(company_name, city)
manages(employee_name, manager_name)

(i) Find the names of all employees who work for First Bank Corporation.

(ii) Find all employees in the database who live in the same cities as the companies for which they work.

(iii) Find all employees in the database who live in the same cities and on the same streets as do their managers.

(iv) Find all employees who earn more than the average salary of all employees of their company.

(v) Find the company that has the smallest payroll.

(vi) Give all employees of First Bank Corporation a 10 percent raise.

(vii) Find the company that has the most employees.

(viii) Find the employees who earn highest salary.

(ix) Find all employees who earn more than the average salary of all.

(x) Create a new table '*employer*' with the attributes *employer_id, employee_name, company_name*, where primary key is (*employer_id, company_name*) and foreign key is *employee_name*.

b) Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$, with primary keys A, C, and E respectively. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of ($r_1 \bowtie r_2 \bowtie r_3$) and give an efficient strategy for computing the join. [2]

(i) Find the names of all employees who work for First Bank Corporation.

Sql

```sql
SELECT employee_name
FROM works
WHERE company_name = 'First Bank Corporation';
```

(ii) Find all employees in the database who live in the same cities as the companies for which they work.

Sql

```sql
SELECT e.employee_name
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
JOIN company c ON w.company_name = c.company_name
WHERE e.city = c.city;
```

(iii) Find all employees in the database who live in the same cities and on the same streets as do their managers.

Sql

```sql
SELECT e.employee_name
FROM employee e
JOIN manages m ON e.employee_name = m.employee_name
JOIN employee em ON m.manager_name = em.employee_name
WHERE e.city = em.city AND e.street = em.street;
```

(iv) Find all employees who earn more than the average salary of all employees of their company.

Sql

```sql
SELECT w.employee_name
FROM works w
JOIN (
    SELECT company_name, AVG(salary) AS avg_salary
    FROM works
    GROUP BY company_name
) avg_salary ON w.company_name = avg_salary.company_name
WHERE w.salary > avg_salary.avg_salary;
```

(v) Find the company that has the smallest payroll.

Sql

```sql
SELECT company_name
FROM works
GROUP BY company_name
ORDER BY SUM(salary) ASC
LIMIT 1;
```

(vi) Give all employees of First Bank Corporation a 10 percent raise.

```sql
UPDATE works
SET salary = salary * 1.10
WHERE company_name = 'First Bank Corporation';
```

(vii) Find the company that has the most employees.

```sql
SELECT company_name
FROM works
GROUP BY company_name
ORDER BY COUNT(employee_name) DESC
LIMIT 1;
```

(viii) Find the employees who earn the highest salary.

```sql
SELECT employee_name
FROM works
WHERE salary = (SELECT MAX(salary) FROM works);
```

(ix) Find all employees who earn more than the average salary of all.

```sql
SELECT employee_name
FROM works
WHERE salary > (SELECT AVG(salary) FROM works);
```

(x) Create a new table `employer` with the attributes `employer_id`, `employee_name`, and `company_name`, where the primary key is `(employer_id, company_name)` and `employee_name` is a foreign key referencing `employee`.

```sql
CREATE TABLE employer (
    employer_id INT,
    employee_name CHAR(20),
    company_name CHAR(20),
    PRIMARY KEY (employer_id, company_name),
    FOREIGN KEY (employee_name) REFERENCES employee(employee_name)
);
```

b) 'To efficiently estimate and compute the join size of :

1. Estimate Size:

First, join with on . This join yields 1500 tuples (since has 1500 tuples).

Then, join the result with on , yielding a maximum of 1000 tuples (matching the primary key size of).

Estimated size of : 1000 tuples.

2. Efficient Strategy:

Perform first to minimize intermediate results, then join the result with .

5. a) Consider the following relations: [10]

client (client-no, name, address, city)
product (product-no, description, profit-percent, qty-in-hand, reorder-level, cost-price)
salesman (salesman-no, name, address, city, sale-amt)
salesorder (order-no, order-date, client-no, del-add, salesman-no, del-date, order-status)
order-detail (order-no, product-no, qty-ordered, qty-delivered)

Give SQL and RA expressions for the following queries:

(i) Find the list of all clients who stay in cities Dhaka or Khulna.
(ii) Find the products with their description whose selling price is greater than 2000 and less than or equal to 5000. [**Hints**: Selling price can be found from cost-price and profit-percent]
(iii) Find the total ordered and delivered quantity for each product with a product range of P0035 to P0056.
(iv) Find the clients with their names and order numbers whose orders are handled by the salesman Mr. X.
(v) Find the product no and description of non-moving products, i.e., products not being sold.

don't know

b) Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$, with primary keys A, C, and E respectively. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $(r_1 \bowtie r_2 \bowtie r_3)$ and give an efficient strategy for computing the join [2]

(i) List of all clients who stay in cities "Dhaka" or "Khulna"

SQL

```sql
SELECT name, city
FROM client
WHERE city IN ('Dhaka', 'Khulna');
```

Relational Algebra

$$\pi_{\text{name, city}}(\sigma_{\text{city}='Dhaka' \text{ OR city}='Khulna'}(\text{client}))$$

(ii) Find products with their descriptions whose selling price is greater than 2000 and less than or equal to 5000

Selling price is calculated as:

$$\text{selling price} = \text{cost-price} + (\text{cost-price} \times \frac{\text{profit-percent}}{100})$$

SQL

```sql
SELECT product-no, description
FROM product
WHERE (cost-price + (cost-price * profit-percent / 100)) > 2000
  AND (cost-price + (cost-price * profit-percent / 100)) <= 5000;
```

Relational Algebra

Letting
$\text{selling price} = \text{cost-price} + (\text{cost-price} \times \frac{\text{profit-percent}}{100})$:

$$\pi_{\text{product-no, description}}(\sigma_{2000<\text{selling price}\leq 5000}(\text{product}))$$

(iii) Total ordered and delivered quantity for each product with product numbers between "P0035" and "P00S6"

SQL

```sql
SELECT product-no,
       SUM(qty-ordered) AS total_ordered,
       SUM(qty-delivered) AS total_delivered
FROM order-detail
WHERE product-no BETWEEN 'P0035' AND 'P00S6'
GROUP BY product-no;
```

Relational Algebra

$$\gamma_{\text{product-no,SUM(qty-ordered),SUM(qty-delivered)}}(\sigma_{\text{product-no}\geq'P0035' \text{ AND product-no}\leq'P00S6'}(\text{order-detail}))$$

(iv) Clients with their names and order numbers whose orders are handled by the salesman "Mr. X"

SQL

```sql
SELECT c.name, s.order-no
FROM client c
JOIN salesorder s ON c.client-no = s.client-no
JOIN salesman sm ON s.salesman-no = sm.salesman-no
WHERE sm.name = 'Mr. X';
```

Relational Algebra

$$\pi_{\text{c.name, s.order-no}}(\sigma_{\text{sm.name}='Mr.X'}(\text{client} \bowtie \text{salesorder} \bowtie \text{salesman}))$$

(v) Product number and description of non-moving products (products not being sold)

SQL

```sql
SELECT p.product-no, p.description
FROM product p
LEFT JOIN order-detail od ON p.product-no = od.product-no
WHERE od.product-no IS NULL;
```

Relational Algebra

$$\pi_{\text{product-no, description}}(\text{product} \setminus (\pi_{\text{product-no}}(\text{order-detail})))$$