Database Systems

Database

collection of related data

- **Structured**
  - implemented using RDBMS
  - stored in the form of table (relation)
- **Unstructured**
  - no predefined structure

DBMS

collection of operations to perform on data like insertion, deletion and updation

- SQL Server
- Oracle 9i, 11, 12c etc
- My SQL
- DB2.

→ File System Vs DBMS

① attributes — searching is easy

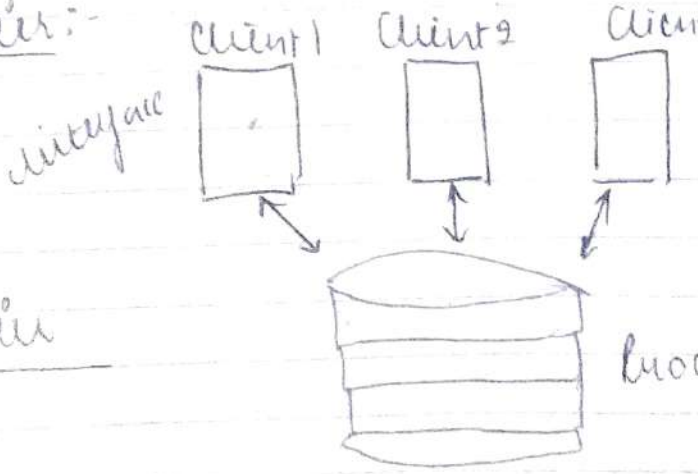
② Concurrency — at same time different users access
- RR
- RW
- WR
- WW

③ Security
- Role based security — different roles to access data
- ⇓
- user based access control

④ Data redundancy :-
- ⇓
- duplicacy

→ Architecture

① 2-tier :-



client server architecture

Eg → Indian Railways

→ Bank records are stored in data in flexible

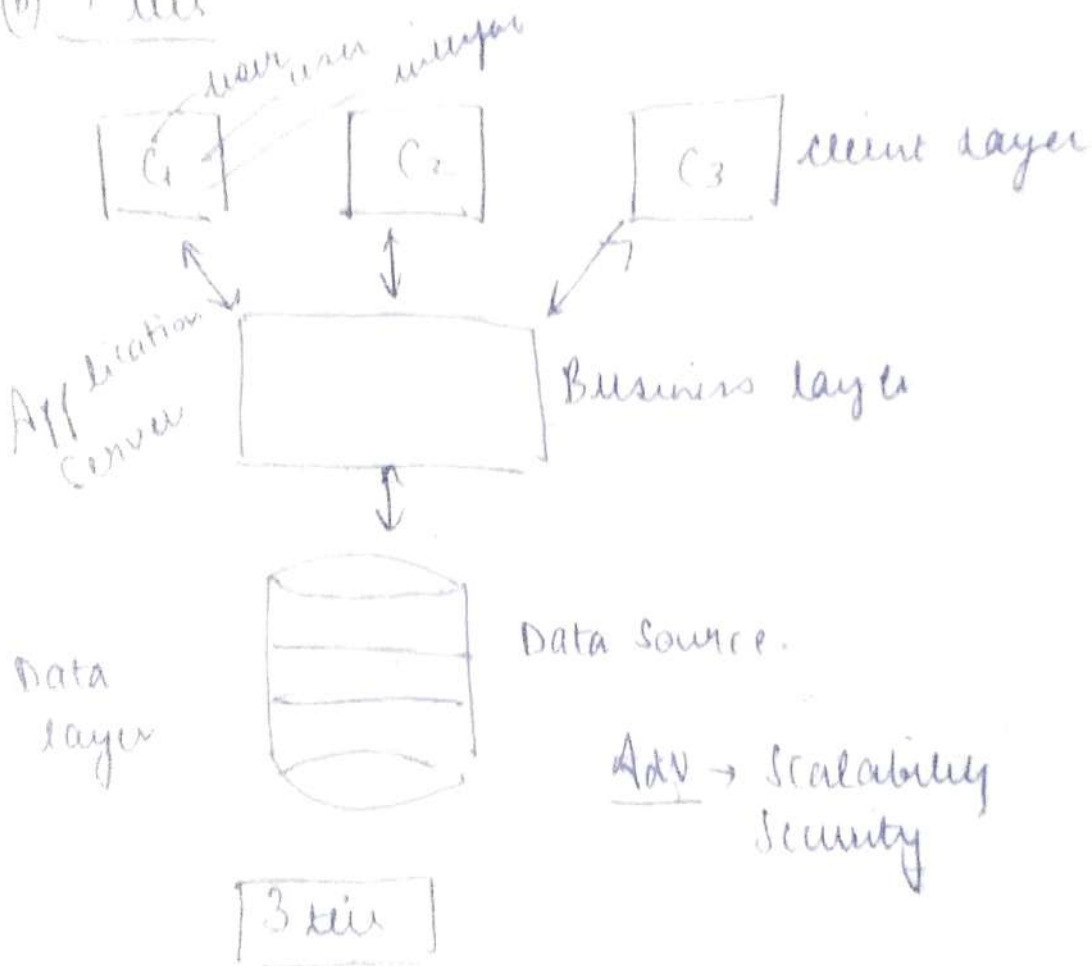
Adv : limited and authorized clients

Disadv : users are not limited (scalability)

Thus (2) maintainance

(3) Security : client is directly interacting with the database

(b) 3 tier



Adv → Scalability
Security

Query after being connected to low level lang is sent to the data layer Thus the security of data layer lessened.

Application server processes the query of

user interacts with data layer using application layer

disadv → complex and maintainance because of presence of application layer

→ Schema :-

logical representation of the database

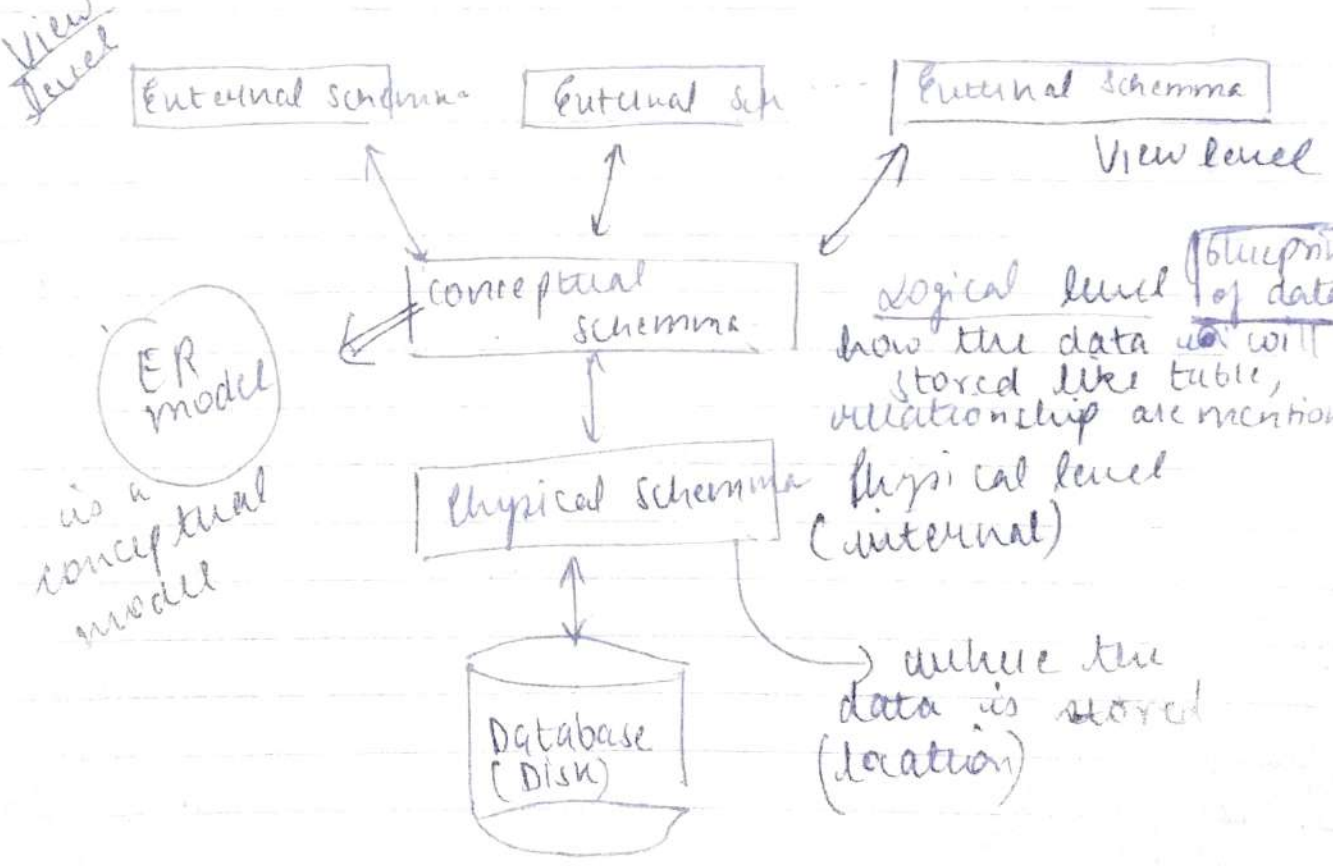
implemented using SQL (DDL)

↳ Data definition language

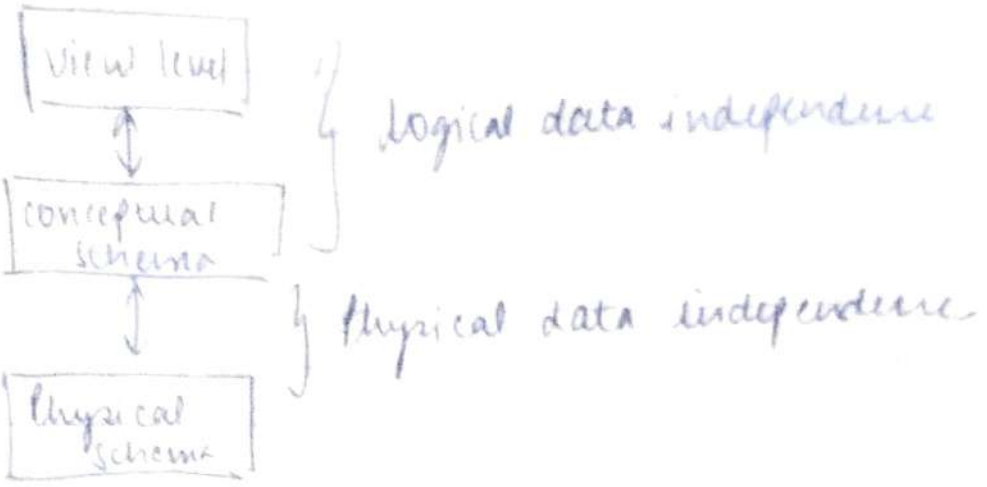
→ Three Schema Architecture –

Schema means structure

Data Independence → user and data donot interact directly. User wont know where the data is stored



→ Data Independence



Type of Keys

Key → attribute
Why → uniquely identify

(a) Candidate Key
Values that are unique and won't repeat

(b) Primary :-
choosen from the set of primary keys
unique + not null

PK

(c) foreign key -
attribute / set of attributes that
refers to primary key of
same table or another table

ER Models

Entity Relation (ER) model is a high level conceptual data model diagram.
ER model helps to analyze data requirements systematically to produce a well designed database.
ER diagram displays the relationships of entity stored in database.

→ Components of ER diagram

(a) Entities :-
  (a) An entity is real world object that can be easily identifiable.
  (b) It can be abstract
  (c) ~~An entity is an object that exists and is distinguishable from other objects~~
  (d) It is distinct
  (e) denoted by
  (f) Eg :- Person : Employee, Student, Patient

(b) Attribute
  (1) It gives characteristics of the entity
  (2) It is also called data element, data field, data item or an elementary item
  (3) It is denoted by  (4) Eg: A person may have attributes: name, age, etc

(c) Relationship :-
  (1) Relationship is nothing but an association among two or more entities
  (2) Eg :- Tom works in Maths by department

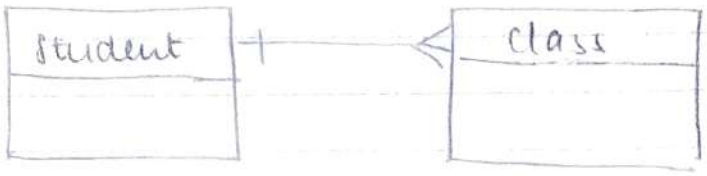
(3) denoted by

② weak entity diff from rest

① Entity Set

① It is a collection of similar type of entities
② it may contain entities with attribute sharing similar value

Eg :- A student entity may have a name, age, class as attributes

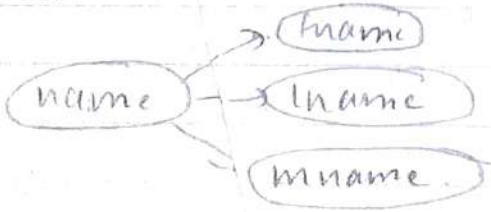


→ Types of Attributes

① Simple Attribute → simple attributes are atomic values which cannot be divided any further. Eg - a student's contact no.

② Composite attribute :- Composite attribute are made of more than one simple attribute. It is possible to break down composite attribute.

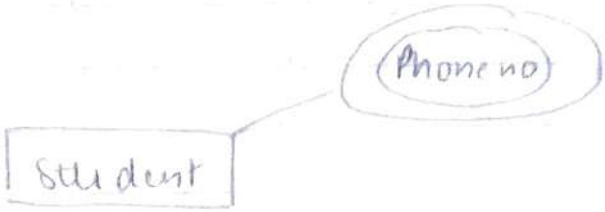
Eg :- a student name might have first name, middle name and last name.



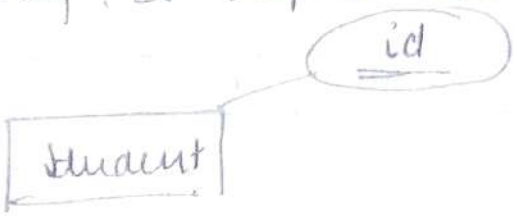
③ Derived Attribute :- Derived attributes do not exist in database. Their values are derived from other attribute that are present in the database.

For eg :- age is derived from DOB

④ Multivalue attribute – Multivalued attributes may contain more than one values
for Eg:- A student can have more than one phone nos



⑤ Key Attribute :-
It is used to represent the main characteristic of an entity. It represents a primary key.
Eg



⑥ Single Attribute
Single value attributes contain single value
Eg → social_security_no

→ Purpose of ER diagram :-
① Database engineer gain a better understanding of the info contained in database with the help of ER diagram
② It is useful to communicate the logical structure of the database to end user
③ used to represent the overall logical structure of the database
④ blueprint for implementing data in specific software applications
⑤ Helps to describe entities, attributes & relationships

①

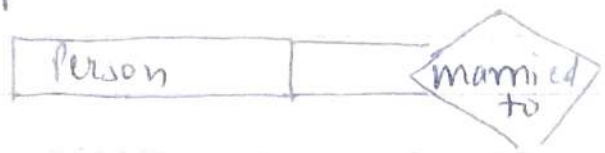
→ Degree of an entity relation type :-

The no of different entity set participating in a relationship set is called as degree of relationship set.

ⓐ Unary Relationship

When there is only one entity set participating in a relation

Eg → one person is married to only one person



ⓑ Binary Relationship

Two entities participating in a relationship

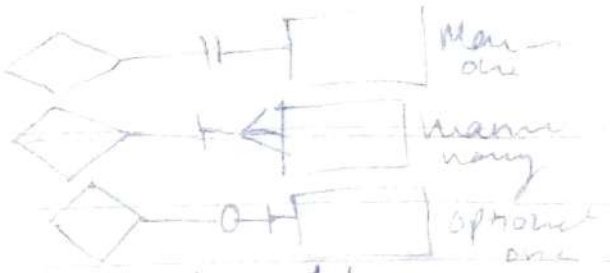
Eg → Student enrolled in course

ⓒ N-ary Relationship

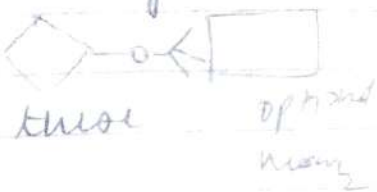
When there are n ~~relationships~~ entities participating in a relationship

→ Cardinality

→ Mapping Constraints :-



(1) Mapping constraints act as rule followed by the data of database.



(2) Data in the database must follow these constraints

(a) Degree of an entity relationship

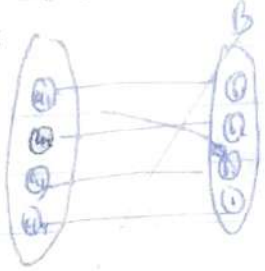
(b) Mapping Cardinalities

Cardinality defines the no of entities in one entity set which can be associated with the no of entities of other set via relationship set

(i) One to one -

One to one (1:1) relationship is when at most one instance of an entity A is associated with one instance of entity B

Eg One student can have only one college Id at a time



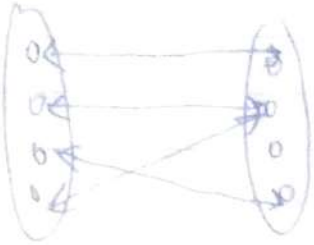
(ii) One to many -

One to many (1:N) relationship is when for one instance of A there are zero, one or many instances of B but for one instance of entity B there is only one instance of entity A.

Eg → a department has many employee but one employee is assigned to one department

(iii) Many to one (N:1)

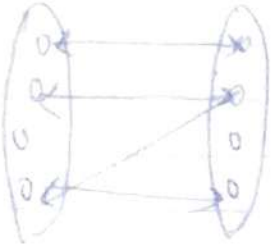
More than one entity set of X can be associated with atmost one entity set of Y. But an entity set Y may or may not be associated with more than one entity from entity set X.



(iv) Many to Many (M:N)

One entity from X can be associated with more than one entity from Y & vice versa

Eg) Students as group are associated with multiple faculty and faculty members can be associated with multiple students

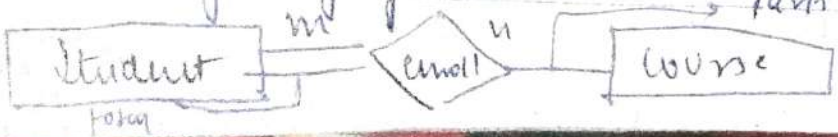


(c) Participation Constraint

It is applied on the entity participating in the relationship

(a) Total → Each entity in the entity set must participate in the relationship Eg → ie each student must enroll in a course

(b) Partial → The entity in the entity set may or may not participate in the relationship Eg → ie some courses are not enrolled by any of the student



- Steps to create ER diagram

① Entity identification
② Relationship identification
③ Cardinality identification
④ Identify attributes
⑤ Create ERD

→ Keys :-

① Key is a attribute or set of attributes that is used to identify data in entity sets
② Key is defined for unique identification of rows in table.

(a) Primary Key

① It uniquely identifies each record in a table & must never be same for records
② cannot be null
③ It is unique
④ It is a candidate key that is used for unique identification of entities within the table. Eg - Student id, Eid

(b) Super Key

① Super key is defined as a set of attributes within a table that can uniquely identify each record within a table.
② it is a superset of candidate key

Eg → student id, phone no etc

ⓒ Candidate

ⓐ Super Key

① It is an attribute (or set of attributes) that is used to uniquely identify all attributes in a relation

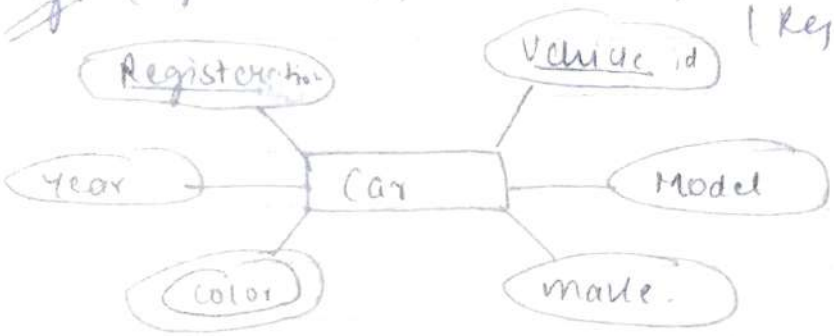
② All super keys cannot be candidate keys

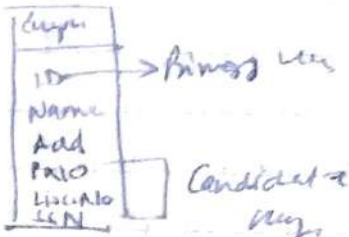
③ It can be null

④ A relation can have any no of super keys

Eg (Registration), (Vehicle id), (Reg id, Veh id)
(Reg id, veh id, Make)

Employee ID (or EmployeeID, Name)





ⓑ Candidate Key

① It is minimal set of super key

② All candidate keys are super keys but not primary keys

③ It can be null

④ No of candidate keys is < no of super keys

⑤ Eg Regid, Vehicle id

ⓒ Primary Key :-

① Primary key is unique and never same for records

② Primary key is a subset of candidate super key

(3) Cannot be null
(4) no of primary key < no of candidate key
Eg → register

(d) Composite key :-

(1) It is a combination of 2 or more columns in a table that can be used to uniquely identify each row in table.
(2) used when we cannot identify a record using single attributes
(3) A primary key that is made by the combination of more than one attribute is called composite key

composite key

stu id	course id	marks	~~score~~ exam

(e) Alternate (Secondary key) :-

(1) Alternate key of any table are those candidate keys which are not currently selected as primary key
(2) One of the candidate key is chosen as primary key & the remaining are called alternate keys

Stude id	Cour id

if student id is primary key then course id is alternate key

⑦ Foreign Key

① Foreign key represents the relationship b/w tables and ensures the referential integrity rule

② A foreign key is derived from the primary key of the same or some other table

③ It can be kept null

Primary key → Foreign key

Eg Suppose employee id in project table points to Employee id in employee table
So employee id in employee → primary
Employee id in project → foreign

→ Extended ER Model

foreign key → primary key

→ Extended ER model

① EER model includes all the concepts of ER model together with following concepts:

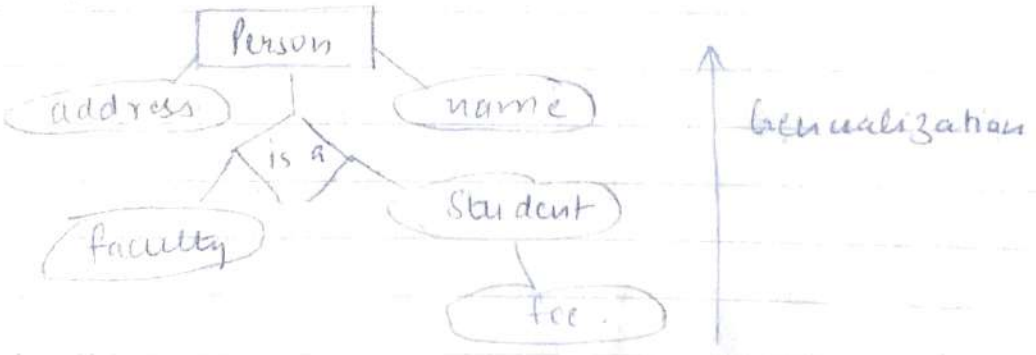
ⓐ Specialization
ⓑ Generalization
ⓒ Aggregation

① Generalization :-

ⓐ bottom up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common

ⓑ It helps in reducing schema size

ⓒ It is the process of extracting common properties from a set of entities & creating a generalized entity from it.

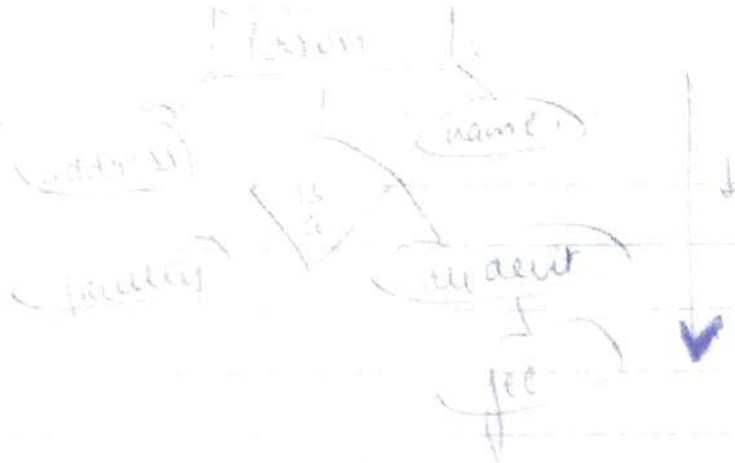


② Specialization :-

① top to down approach. In this higher level entity can be broken into two or more lower level entities

② used to identify the subset of an entity that shares some distinguish characteristics

③ Reduces the size of schema

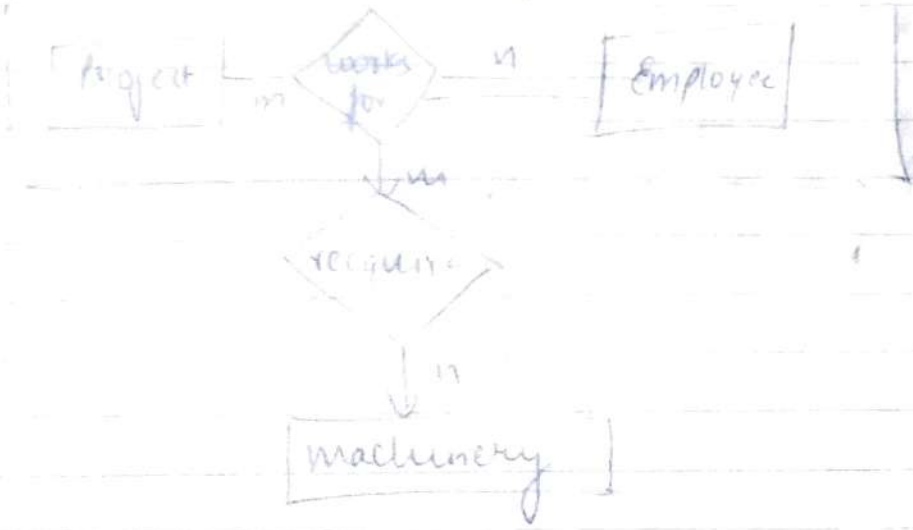


⑥ Aggregation

Aggregation is a process when relation between two entities is treated as single entity

② An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In such cases a relationship with its corresponding entities is aggregated into a higher level entity

③ Increase the size of schemas



Aggregation

③ The superclass/subclass entity types is one of the most included in EER model

④ This help us to model a general entity & then subdivide it into subclasses or subtypes

⑤ A subclass is an entity type that has a distinct role & is also a member of a superclass

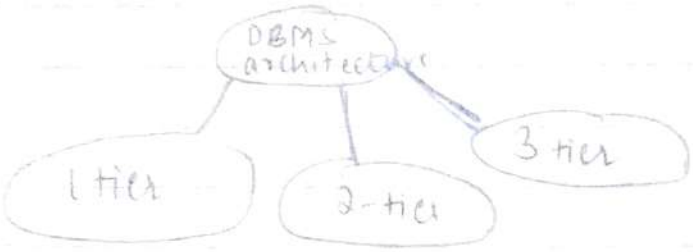
→ Reduction of ER diagram

ⓐ Entity becomes table
ⓑ single value attributes → column
ⓒ key attribute represented by primary key
ⓓ multivalued attribute → ~~with components~~
ⓔ separate table
ⓔ Composite attributes → only components are written
ⓖ Ignore derived attributes

Arrows from [Primary → foreign key]

→ DBMS Architecture –

The DBMS design depends upon its architecture DBMS architecture depends upon how users are connected to the database to get their request done.



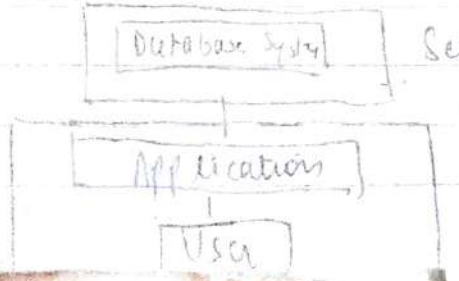
(a) 1 tier –

(a) In this the database is directly available to the user
(b) Any changes done here will be directly done on the database
(c) used for development of local applications

(b)

(b) 2- tier :-

(a) In this application on client end can directly communicate with the database on server end.
(b) The user interfaces and application programs are run on the client side.
(c) Server side is responsible to provide the functionalities like query processing & transaction management.
(d) To communicate with DBMS, client side application establishes a connection with the server side



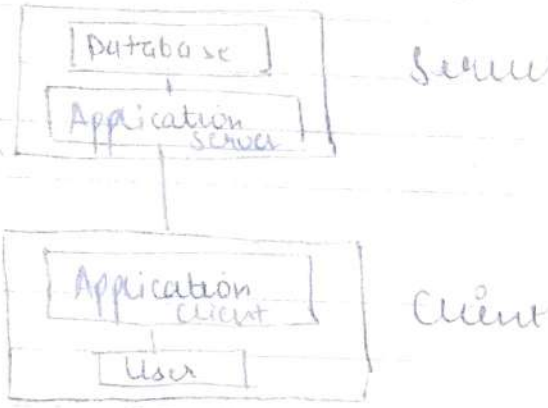
⑥ 3-tier :-

① In this there is a layer between the client & server

② In this the client cannot directly communicate with the server

③ The application on the client end interacts with an application server which communicates with the database.

④ used in case of large web application



→ Data Independence

Data independence refers characteristic of being able to modify schema at one level of the database system without altering the schema at next higher level

ⓐ Logical Independence

① it is mainly concerned with the structure or changing the data definition

② protection from changes in logical structure of data

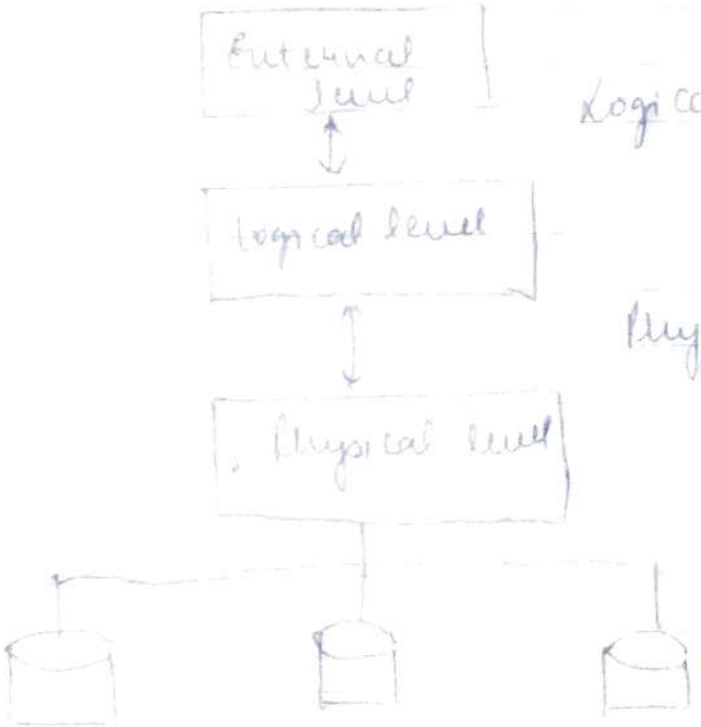
ⓑ Physical Independence

concerned with data storage

→ protection from changes in physical structure of data

- difficult to retrieve data
- it is difficult as the retrieving data is dependent on the logical structure of data
3) we need to make changes in application program if new fields are inserted or deleted
4) concerned with conceptual schemma
eg → add / Modify / delete new attribute

- easy to retrieve physical
2) It is easy to retrieve data
3) It change at physical level usually does not need change at the application program level.
4) concerned with internal schemma
Eg - change in compression technique, hashing etc.



Imp → (a) improve quality of data
(b) Database system becomes affordable
(c) Database incongruity is vastly reduced.
(d) don't need to alter data structure in application programs.

→ Schema

(a) Overall design of database is called schemma

(b) It represents the logical view of the entire database.

(c) It defines how the data is organized & how the relation among them are associated

Three Schema Architecture

The three schema architecture consist of 3 levels :-

(a) Internal level -

1) Internal schemma is also known as physical schema

2) Used to define how the data will be stored in a block.

3) Internal level has an internal schema that describes physical storage of database

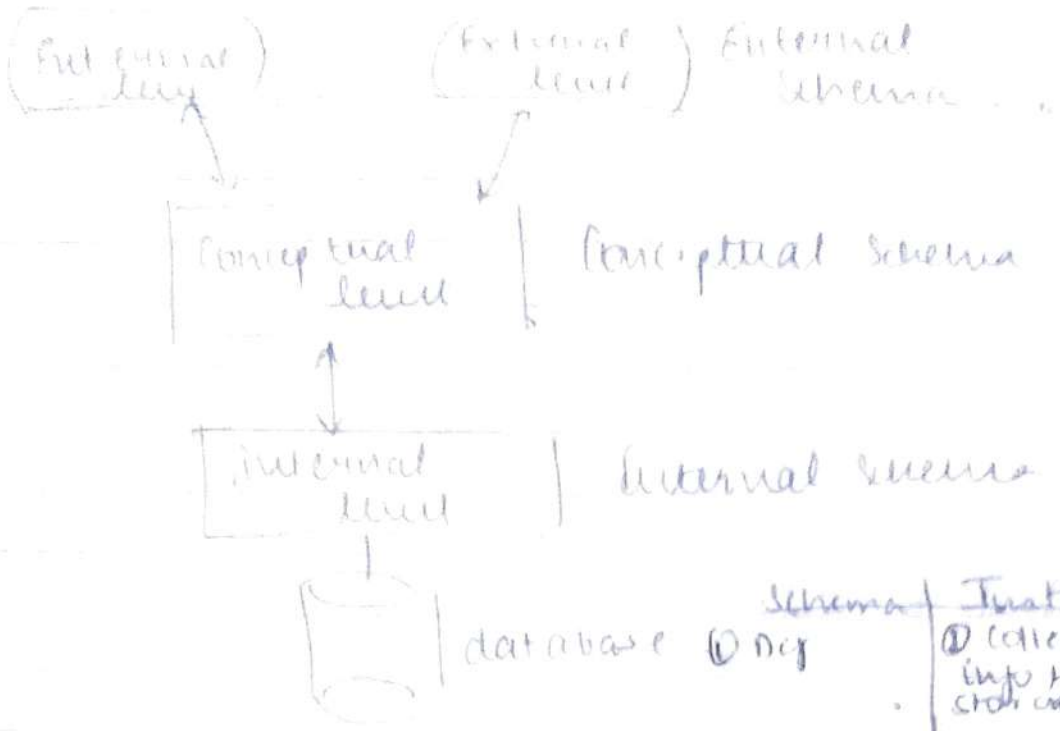
(b) Conceptual level :-

1. describes the design of database at the conceptual level (logical level)
2. it describes what data will be stored in the database and relationship among existing data
3. In this internal details are hidden

(c) External level :-

1. External schemma also known as view schemma
2. describes the end user interaction with the database

It describes how the data in which user is interested & hides the remaining data from



Schema
① Definition of the database

② can't be changed

Instance
① Collection of info that is stored in db at a specific moment

can be changed

→ Database Instance :-

Database Schema is the skeleton of the database. It is designed when the database doesn't exist at all. Once the database is operational it is hard to make changes

A database instance is a state of operational database with data at any given time. It tends to change with time

→ DBMS Interfaces :-

A DBMS interface is a user interface that allows the ability to input queries to a database without using the query language itself

CIMNFS

(from ppt)

③ tells us how the data will be stored

~~text two~~ is the set of attributes that is stored at a particular time

→ DBMS languages :-

(from quantum + notes types)

(Q/)

Sid	Cid
S_1	C_1
S_2	C_1
S_1	C_2
S_3	C_2

Enrolled

Cid
C_1
C_2

Output → S_1

→ Basic SQL Relational Algebra Operations

① Unary Relational Operations :-

Select (σ)

used for selecting subset of tuples according to a given condition

σ (r)

- predicate → prepositional logic
- r → relation i.e the name of table

Project (∏)

helps us to keep specific columns from a relation & discard other columns.

Rename (ρ)

used to rename a relationship or an attribute

Syntax

ρ (new rel name, old relation name)

ρ (cust_name, ∏ (cust_name) (customer))

② Relational Algebra Operations from Set theory :-

(a) Union (∪)

used to select all the rows from two tables and duplicates

tablename1 ∪ tablename2

(b) Intersection (∩)

used to select common rows from 2 tables

table1 ∩ table2

(c) Set Difference (−)

We have two relations R1 & R2 we want to select tuples that are present in R1 but not present in R2 then we do R1 − R2

table name 1 − table name 2

④ Cartesian product :- (X)

merge two columns from two relations.

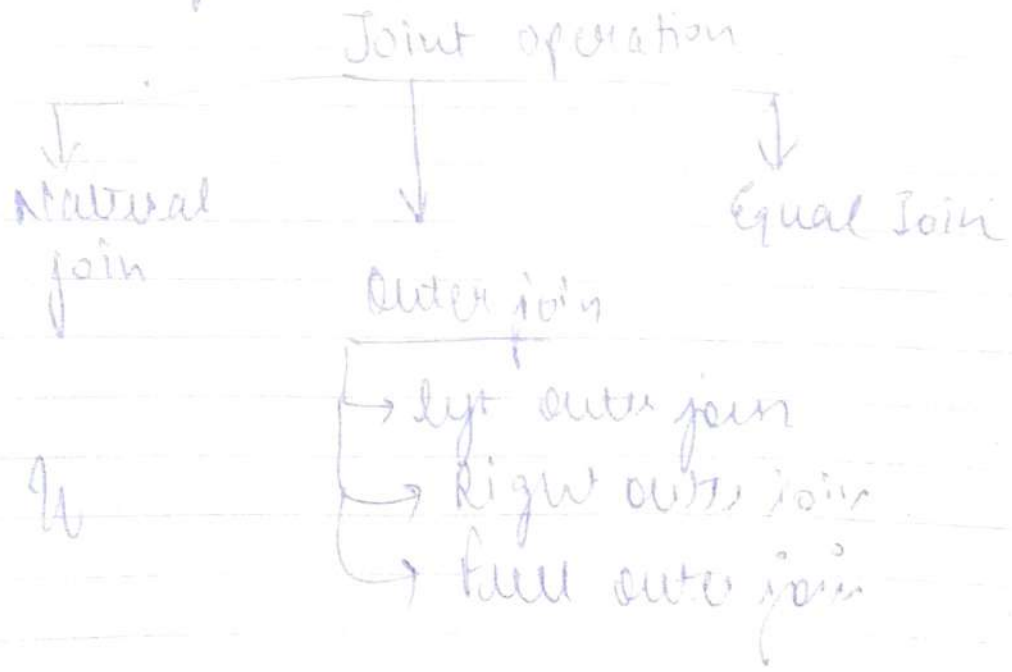
r X s

⑤ Binary Relational Operations

→ Hence we have to use 2 tables

ⓐ Joint operation (⋈) :-

It combines related tuples from different relations if join condition is satisfied.



ⓐ ~~Natural Join~~ (⋈)

ⓐ Join Operation :-

① There must be some common attribute between 2 table
② Cross product + Select statement (some condⁿ)

① Natural Join :- (⋈) Set of all tuples of all combinations in R & S that are equal on their common attribute names

Eg

Emp

ENO	Ename	Address
1	Ram	Delhi
2	Varun	Chd
3	Ravi	Chd
4	Ankit	Delhi

Dept

Dept No	Name	ENO
D1	HR	1
D2	IT	2
D3	MRKT	4

Ques) Find the employee name who is working in a department

Select Empname from Emp, Dept where Emp.Empno = Dept.Eno;

Soln ① Cross Product → no of rows = 4 * 3 = 12
no of columns = 6

ENO	Ename	Address	Dept No	name	Eno	
1	Ram	Delhi	D1	HR	1	
1	Ram	Delhi	D2	IT	2	X
1	Ram	Delhi	D3	MRKT	4	X
2	Varun	Chd	D1	HR	1	X
2	Varun	Chd	D2	IT	2	✓
2	Varun	Chd	D3	MRKT	4	X

1	Ram	Delhi	D1	HR	
2	Varun	Und	D2	IT	2
4	Amrit	Delhi	D3	MKT	4

We have to print only ename so

Ram Varun Amrit

⇒ output

* Ravi is rejected bcz no department

Select Ename from Emp Natural Joint Dept

Cross Product / Cartesian Product → merge columns from 2 relations

total columns = $m + n = 3 + 3 = 6$

total tuples = $p * q = 2 * 2 = 4$

Eg

R1 =

A	B	C
1	2	3
2	1	4

R2

C	D	E
3	4	5
2	1	2

R1 × R2

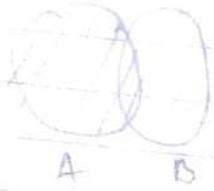
A	B	C	C	D	E
1	2	3	3	4	5
1	2	3	2	1	2
2	1	4	3	4	5
2	1	4	2	1	2

(b) Outer Join :-

Used to deal with missing information

(a) Left Outer Join ⟕

It gives the matching rows & the rows which are in left table but not in right table.



Emp

Emp no	Ename	Dept no
E1	Varun	D1
E2	Amrit	D2
E3	Ravi	D1
E4	Nitin	—

Dept

Dept No	Dname	Loc
D1	IT	Delhi
D2	HR	Hyd
D3	Finance	Pune

Input - select emp no, ename, dname, loc
from emp left outer join dept on
(emp dept no = dept dept no)

Output

Eno	Ename	Dname Dept no	Loc
E1	Varun	D1 IT	Delhi
E2	Amrit	D2 HR	Hyd
E3	Ravi	D1 IT	Delhi
E4	Nitin	—	—

(b) Right Outer Join :-

It gives matching rows & the rows which are in right table but no in the left table.

Emp

Emp No	Ename	Dept no
E1	Varun	D1
E2	Amit	D2
E3	Ravi	D3

Dept

Dept No	Dname	Loc
D1	IT	Delhi
D2	HR	Hyd
D3	Finance	Pune
D4	Testing	Noida

Input → Select Empno, dname, loc from
Emp Right Outer joint dept On
(Emp deptno = dept dept no)

Output

Emp No	dname	Loc
E1	[illegible] D1	Delhi
E2	[illegible] D2	Hyd
E3	D3	Pune
	D4	Noida

null

(c) Full Outer Join

Full outer join is like left or right join except that it contains all rows from both tables

→ Equi Join :- (=)
(Inner Join)
based on matched data as per the equality condition

Emp

ENO	EName	Address
1	Ram	Delhi
2	Varun	Chd
3	Ravi	Chd
4	Amit	Delhi

Dept

DeptNo	Loc	Eno
D1	Delhi	1
D2	Pune	2
D3	Patna	4

1	Ram	Delhi	D1	Delhi	1	✓
1	Ram	Delhi	D2	Pune	2	✗
1	Ram	Delhi	D3	Patna	4	✗
2	Varun	Chd	D1	Delhi	1	✗
2	Varun	Chd	D2	Pune	2	✗
2	Varun	Chd	D3	Patna	4	✗

Input :- Select Ename from Emp, Dept where
Emp.Eno = Dept.Empno and
Emp.address = Dept.location

Output

Ram

→ DBMS language :-

Data Definition language is used by Data Base Administrator an designers to define both conceptual & internal schemas.

Storage Definition language (SDL) is used to specify internal schema

View definition lang, used to specify user views & their mapping to the conceptual schema.

Data Manipulation language (DML) Retrieval, insertion, deletion & modification of the data is done using DML.

Two types :-

(a) Non procedural DML (high level) used to specify complex database operations concisely.

(b) Low level or Procedural are called record at a time DMLs. They retrieve individual records or objects from the db & process each separately

Delete	Truncate
It delete rows of the table when condⁿ is specified & deletes all rows if the condⁿ is not satisfied	used to delete all rows from table free the space
Drop	the structure of table is not deleted
deletes the entire structure of table	

(8) Alter

(a) Rename →

alter table tablename rename to newtablename;

(b) Truncate →

truncate table tablename;

(c) add →

alter table tablename add (column name datatype);

(d) Modify →

alter table table_name modify columnname datatype;

(e) drop -

alter table tablename drop column columnname;

(f) Rename column

alter table tablename rename column oldname to newname;

(9) Select -

select expression from
database where cond";

(10) Select Distinct -

used to select retrieve only
distinct data

Select distinct columnname, columnname
from table name;

(11) Select Count :-

used to return no of rows
in the query

Select count(exp) from table name
where cond";

Select count(*) from table name;
to return no of records in table

(12) Select count (distinct name) from
table_name;

(12) Select top

top data from table

Select top 2 * from table name;

(13) Where - (DML)

Where clause like

=
>=
<=
<
>
<> not equal to

(14) Order by :-

Select * (exp) from tablename where cond^{n} order by exp [ASC | DESC];

Eg select * from xyz order by name, salary, colm;

→ Set Operations

(15) Union

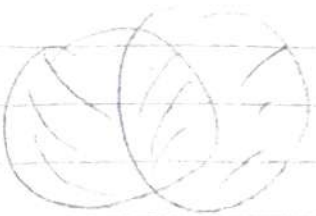
select * first
union
select * second

combine two or more select statement union out duplicate

(16) Union all :-

same as union show duplicate value

select * 1st
union all
select * 2nd;



(17) Intersect

common in both selects

```
Select * from A
intersect
Select * from B;
```

(18) Minus

```
Select * from A
minus
select * from B;
```

→ Aggregate func

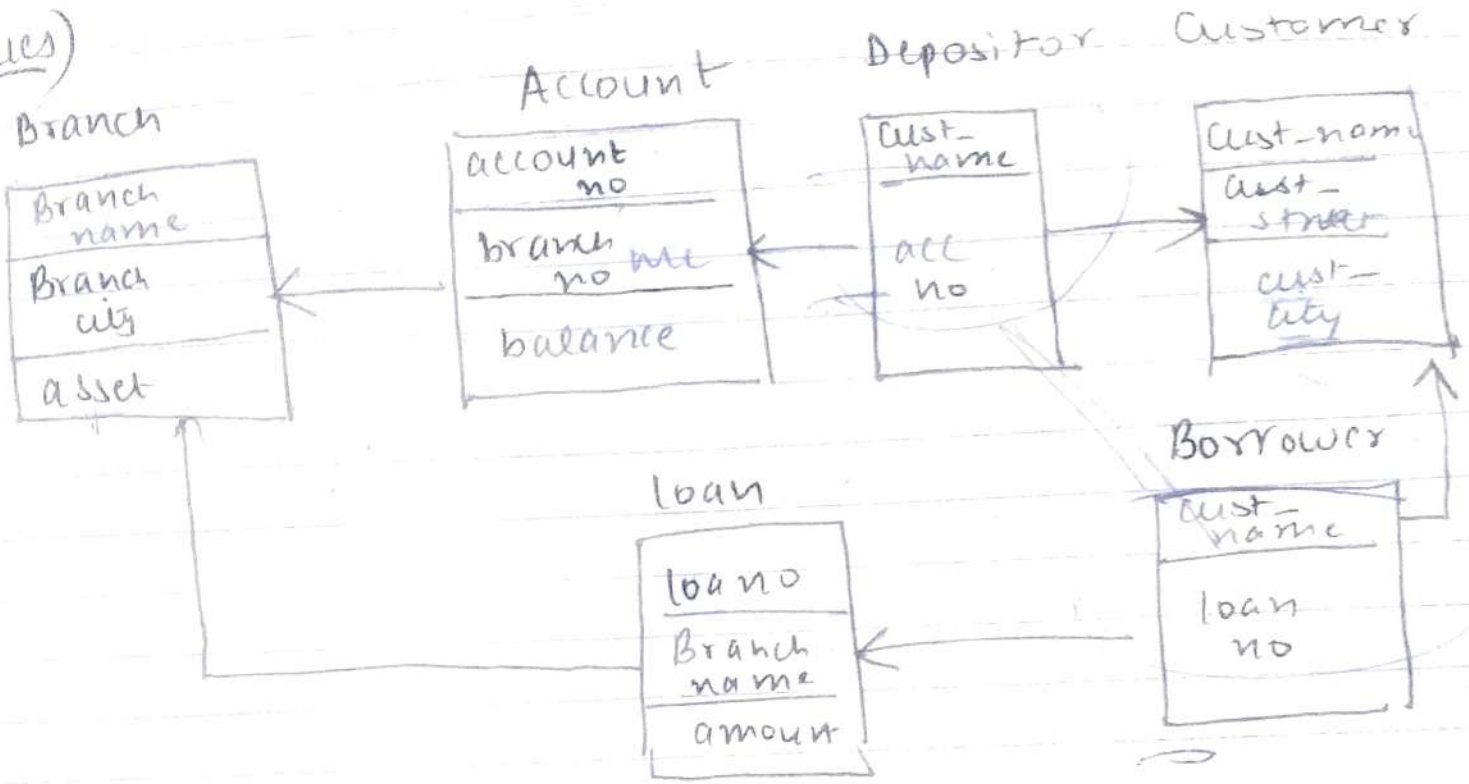
(1) min (2) max (3) count

```
Select * ~~min~~ (column(*)) from
                    table name;
```

(4) Sum (5) Avg

```
Select sum(column) from tablename;
```

(Ques)



(a) find the details of customer who lived in Delhi.

Π_{Customer} ~~$\sigma_{\text{cust-city} = \text{"Delhi"}}$~~

$\sigma_{\text{cust-city} = \text{"Delhi"}}(\text{Customer})$

(b) Find the details of those whose loan amount is having less than 5000.

$\sigma_{\text{amount} < 5000}(\text{loan})$

(c) Branch is non Delhi & loan amount is less than 5000.

$\sigma_{(\text{Branch city} \neq \text{Delhi} \wedge \text{amount} < 5000)}(\text{loan})$

of branch as Delhi or amount as equal to greater than 10000

$$\sigma_{(\text{Branch name} = \text{Delhi} \vee \text{amount} \geq 1000)}(\text{loan})$$

(e) find all the branch name of the bank

$$\Pi_{\text{branch name}}(\text{branch})$$

(f) find the customer name who have a loan

$$\Pi_{\text{cust\_name}}(\text{Borrower})$$

(g) find those account numbers where the balance is less than 1000

$$\Pi_{\text{account no}}(\sigma_{\text{balance} < 1000})(\text{account})$$

(h) find customer name who has loan

$$\Pi_{\text{custname}}(\text{borrower})$$

(i) Find those loan no. which are from Delhi branch with amt > 1000

$$\Pi_{\text{loan no}}\left(\sigma_{\substack{\text{branch}\\ \text{name}=\text{Delhi}}} \wedge \text{amt} > 1000\right)(\text{loan})$$

(j) Find the branch name and branch city where asset is more than 1lakh.

$$\Pi_{\substack{\text{branch name},\\ \text{branch city}}}(\sigma_{\text{asset} > 1\text{lakh}})(\text{Branch})$$

(k) Find the name of the customer who has loan (or) account (or) both.

$$\Pi_{\text{cust\_name}}(\text{borrower}) \cup \Pi_{\text{cust\_name}}(\text{Depositor})$$

(l) Find the name of the branch that has account but no loan.

$$\Pi_{\text{branch name}}(\text{account}) - \Pi_{\substack{\text{branch}\\ \text{name}}}(\text{loan})$$

(m) Find the customer name that has neither loan nor account

$$\Pi_{\substack{\text{customer}\\ \text{name}}}(\text{customer}) - \left(\Pi_{\substack{\text{custo}\\ \text{name}}}(\text{borrower}) \cup \Pi_{\text{cust\_name}}(\text{Depositor})\right)$$

(h) Find the customer name having account balance < 1000

$$\Pi_{cust\_name}\left(\sigma_{bal<1000}\left(account \times deposit\right)\right)$$

Relational Calculus

Relational Calculus

- Tuple relational calculus
  - calculus on tuple
  - $\{t \mid P(t)\}$ (P(t) = condition)
  - t = resulting tuples
  - P(t) = Predicate & these are conditions that are used to fetch t
  - ↳ OR (∨)
  - ↳ AND (∧)
  - ↳ NOT (¬)
  - ↳ there exists $\exists t \in r(Q(t))$ ↳ there exists a tuple t in relation r such that predicate Q(t) is true
  - ↳ for all $\forall t \in R(Q(t))$ ↓ Q(t) is true for all tuples in relation r

- Domain Relational Calculus
  - calculus on attribute
  - Records are filtered based on the domains

2022/6/6

Schemma of Bank

branch (branchname, balance, assets)
customer (cname, cstreet, ccity)
account (Ano, branch, balance)
loan (loanno, branchname, amount)
depositor (cname, accno)
borrower (cname, loanno)

(1) find the name of all customers who have a loan at the bank along with loan number and loan amount.

$$\Pi_{cname}\left(\sigma_{\substack{borrower.no \\ = loan.no}}(loan \times borrower)\right)$$

Using natural join :-

$$\Pi_{cname}(borrower \bowtie loan)$$

Ques) Find the name of all branches with customer who have an account in bank & who live in "Harrison".

$$\Pi_{branchname}\left(\sigma_{\substack{customer \\ city = "Harrison"}}(customer \bowtie depositor \bowtie Account)\right)$$

2022/6/5

Find all the customer who have a loan & account at the bank

Π_{cname} (borrower ⋈ depositor)

Ques) Find the name of all customers who have a loan at Delhi branch

~~Π_{cname} (customer~~

Π_{cname} ($\sigma_{branch\ name = "Delhi"}$ (loan x branch))

Ques) Find the name of all the customers who have a loan at Delhi branch but don't have an account at any branch

$\Pi_{custname}$ ($\sigma_{branch\ name = "Delhi}$ (loan x branch)) −

$\Pi_{cust\ name}$ (depositor))

2022/6/6

Ques) Schemma of Bank

branch (bname, bcity, assets)
customer (cname, cstreet, city)
account (accno, bname, balance)
loan (loanno, bname, amount)
depositor (cname, accno)
borrower (cname, loanno.)

Q1) find the loan no, branch name & amount of all loan of over 1200.

Relational algebra: $\Pi_{\text{loanno, branch name, amount}}(\sigma_{\text{amt}>1200}(\text{loan}))$

Relational calculus:

$\{t.\text{loanno}, t.\text{branchname}, t.\text{amount} \mid \text{loan}(t) \text{ AND } t.\text{amount} > 1200\}$

Q2) find the loan no for each loan of amount greater than 1200.

~~$\{t.\text{loanno} \mid \text{loan}(t) \wedge \text{amount} > 1200\}$~~

$\{t.\text{loanno} \mid \text{loan}(t)$

2022/6/

(3) Find the name of all customers having a loan at Delhi branch.

~~{t.name | borrower(t) ∧ borrower(s)}~~

{t. cname | borrower(t)
∧ (∃l)(loan(l) ∧ l.bname =
"Delhi" ∧ l.loanno = t.loanno}

(4) Find the name of all the ~~loan & account~~ customers having loan & account.

{t.cname | borrower(t) ∧
∃((∃l)(∃a)(loan(l) ∧ account(a))
∧ (t.loanno = l.loanno) ∧
(
{t.cname

DRC

(a) Find loan no, branchname & amount of all loan of over 1200.

$\{ \langle loanno, branchname, amount \rangle \mid \in loan \wedge amount \geq 1200 \}$

(b) Find the loan no for each loan of amount > 1200

$\{ \langle loanno \rangle \mid \in loan \wedge amount > 1200 \}$

(c) Find the name of all customers having a loan at Delhi branch.

$\{ \langle cname \rangle \mid \in borrower$

$\langle cname, lno \rangle \in borrower.$

$\exists\, l\text{-}no, bname (\langle cname, lno \rangle \in b \wedge \langle l\text{-}no\text{-}b\text{-}name, amount \rangle \in loan \wedge bname = "Delhi") \}$

(a) $\{ \langle loanno, branchname, amount \rangle \mid$

$\langle loanno, branaw, amount \rangle \in$

$loan \wedge amount > 1200 \}$

(b) $\{ \langle loanno \rangle \mid$

$\exists (bname, amount)$

$(\langle loano, \rangle \in loan$

$brana, 'eu$

$\wedge$ ~~bname = 'Delhi~~ $amount > 1200 \}$

(c) $\{ \langle cname \rangle \mid \exists loanno, bname, amount$

$\langle cname, loan\ no \rangle \in borrower$

$\wedge \langle loanno, bname, amount \rangle$

$\in loan \wedge amount > 1200 \}$

(d) find the name of all the customers having loan and account.

$\{ \langle cname \rangle \mid$

→ Group by :-

To create groups eg. find out no of customers in each country

means - similar data in groups

Group by statement is often used with aggregate function.

Group by clause used with select

Always after where, before, order by

Syntax :-

select columnname, aggregate fun" (columnname) from tablename (where condition) Group by columnname ;

where → filtering at individual level

→ Having :-

~~It is~~ used where the

It was added to SQL because 'where' clause keyword cannot be used with aggregate function.

List the no of customer in each country with more than 5 customers.

select country, count(customer_id) from customer Group by country having count(customer) > 5 order by count(customerid) Desc;

2022:6:6

→ View :-

View is a virtual table,

not all superkey

→ Tuple Relation Calculus :-

logicaley

* OR(∨) AND(∧) NOT(¬)

* ∃ t∈R(Q(t))
there exist a t in r Q(t) is true

* ∀t ∈R(Q(t))

Eg ① {T.name | Author (t) and
T. article = "database"}

{T | P(t)} condition

Step at → it returns a tuple with name from author who has written an article on 'database'.

{R | ∃T ∈ Author(T.article = "database")
and R.name = T.name)

Eg)

FN	LN	Age
Pinky	Singh	30
Dolly	singh	31
Tom	Wilg	27
Kuck	Bhap	28

Ques Display the last name of those students whose age > 30

{t.LN | student(t) and t.age > 30}

O/P Singh Singh

Ques) Display all the details of student where last name as "singh".

$\{t.ln \mid$ Student(t) and
lastname = "singh"$\}$

$\{t \mid$ student(t) and
t.ln = "singh"$\}$

O/p ⇒

FN	LN	Age
Pinky	Singh	30
Daly	Singh	30

→ Domain Relational Calculus :-
In DRC the records are filtered based on the domains
It uses same operators as TRC.

Notation →

$\{a_1, a_2, \ldots, a_n \mid P(a_1, a_2 \ldots a_n)\}$

where $a_1, a_2 \ldots$ are attributes
P ⇒ condition for fetching the data

Eg $\{<article, page, subject> \mid$
$\in$ Author $\wedge$ subject = "database"$\}$

eg)

FN	LN	Age
Pinky	Singh	30
Dolly	Singh	31
Tom	King	27
Lucky	Pratap	26

Ques) Find the FN and age of student age is greater than 27.

$\{ < FN, age > \mid \in Student \wedge age > 27 \}$

Output → Pinky ~~Singh~~ 30
Dolly ~~Singh~~ 31
Lucky 26

~~Forei~~ Foreign key reference tablename

→ Subqueries

Emp

Eid	Ename	Dept	Salary
1	Ram	HR	1000
2	Amit	MRKT	2000
3	Ravi	HR	3000
4	Nitin	MRKT	4000
5	Varun	IT	5000

Q1) Write a SQL query to display maximum salary from Emp table.

select max(salary) from Emp

Q2) Write a SQL Query to display Employee name who is taking maximum salary

~~select Emp~~
~~(select max(salary) from Emp~~

select Ename from
Emp where
salary = (select max(salary)
(from Emp));

inner query

2022

Ques) Write a SQL Query to display second highest salary from table.

Select max(salary) from Emp where salary <> select max(salary) from (Emp);

Ques) Write SQL Query to display Employee name who is taking 2nd highest salary.

Select Ename from Emp where salary = (select max(salary) from Emp where salary <> (select max(salary) from Emp));

Ques) Group by

Eid	Ename	Dept	Salary
1	Ram		
2	Amit	HR	
3	Soha	MRKT	
4	Sri	HR	
5		MRKT	
		HR	

Count no of employee in same dept

Select dept, count(*) from Emp group by dept;

Ques) Write a query to display all the dept ~~names~~ no where no of employees are less than 2.

```
select dept from Emp group by dept;
    having count(*) < 2;
```

Output

IT

Ques) Tell Empname in above ques.

~~select~~

```
select Ename from Emp
where dept in ((select dept from
Emp group by dept having
    count(*) < 2;
```

Ques) Write a query to display highest salary department wise & name of Emp who is taking that salary.

~~select from~~

Output → 30000
40000
50000

select dept, max(salary) from Emp
group by dept

to find name :-

select Ename from Emp where
salary in (select max(salary)
from Emp group by dept);

↓↓

Output → Ravi

→ IN / NOT IN

Emp
(Eid Ename Address)

Project
(Eid, Pid, Pname, location)

Ques) Detail of Emp whose address is either Delhi or Chd or Pune

select * from Emp where
~~Address = "Delhi~~
Address (in) ('Delhi', 'Chd', 'Pune');

↓

if not in then apart from
Delhi, Chd, Pune

Ques) find the name of Emp who are working on a project.

~~(select Eid from Project)~~
select Ename from Emp where Eid in
(select distinct(Eid) from Project)
similarly for notin

→ we check inner query again & again
→ Exists/Not Exists :- always return true and false

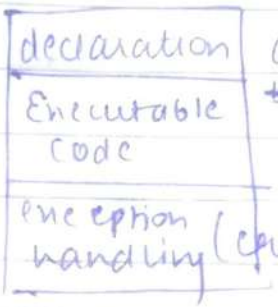
Ques) find the details of Emp who is working on atleast one Project.

select * from Emp where
Eid Exists (select Eid from
Project where Emp.Eid = Project.Eid);

Output → 1, Ravi Chd
3, Nitin, Pune
4, Robin, Pune.

→ PL-SQL
(Procedural SQL)

① function
② procedure
③ Trigger
④ Cursor



sum of 2 nos

```
declare
  a int;
  b int;
  c int;
begin
  a:=10;
  b:=20;
  c:=30;
  c:= a+b;
end;
```

to take input from user

a:=&a

```
dbms_output_put_line("sum of a and b =" ||c);
end;
```

20221

truncate
[table tablename;

Delete	Truncate	Drop
delete specific rows when condition is specified or delete all rows when condition is not specified Delete from tablename [where condn]	delete all rows and free the containing space the table structure remains same • is faster & uses less resources than delete	Entire schema is deleted ↳ table structure is dropped ↳ Relationship will be dropped ↳ Integrity constraint will dropped ↳ access privileges will be dropped

drop table tablename

→ Alter

→ for column
alter table tablename rename column oldname to newname

(a) Rename:-

alter table tablename rename to new tablename;

(b) add column:-

alter table tablename ADD
(columnname1 datatype,
columnname2 datatype);

(c) Modify column

alter table tablename
modify columnname datatype(new size);

(c) Add Constraint

alter tablename tablename add constraint ____ ( )

(d) Drop Column

alter table tablename drop column column name;

Select

ID	Fname	Lname	Age	Subject
1	Amar	Sharma	20	Maths
2	Akbar	Khan	22	Bio
3	Anthony	Milton	23	Comm.

student

(a) Clauses :-

(a) from → select fname from student;

O/p ⇒ Amar
Akbar
Anthony

(b) where :-

(c) with -

(d) Order By - By default ascending

select expression from tablename where cond^n Order By Expression [Asc|DESC];

2022-06-06

(b) Unique / Distinct :-

Both are same

Select unique columnname from table;

Select distinct columnname from table;

(c) Count :-

Select count(name) from Emp;

Select count(*) from Emp;

↙ display no of records.

Select count(distinct name) from Emp;

↙ Count distinct name.

(d) Top :-

Select TOP 2 * from Emp;

20221016

→ UNION

```
select * from first
Union
select * from second;
```

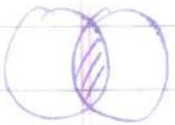
no duplicate values

UNION ALL

```
select * from
first
Union All
select * from
second;
```

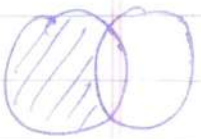
duplicate values

→ Intersect :-



```
select * from first
intersect
select * from second;
```

→ MINUS :-



```
select * from first
minus
select * from second;
```

→ Aliases :-

used to give a table, or a column a temporary name.

```
select customerID as ID, customername
as customer from customers;
```

```
select column name as
alias name from table name;
```

20915

→ INSERT

→ IS NULL NOT NULL

→ Update

update tablename
set ~~contact~~ column1 = value1, column2 = value2,
where condition;

Eg update Customer
set contactname = "Shivangi",
contactno = "985623"
where cid = 1;

→ Like
select from tablename
where column like "_xxxx%.";

2022/6/6

Subquery, view & PSJ

Ques) passenger(pid, pname, pgender, pcity)
agency (aid, aname, acity)
flight (fid, fdate, time, src, dest)
booking(pid, aid, fid, date)

(a) Get Complete details of all flights to New Delhi.

$\sigma_{dest="New\ Delhi"}(flight)$

select * from flight where dest = "New Delhi"

$\{ t \mid flight(t) \wedge t.dest = "New\ Delhi" \}$

$\{ < fid, fdate, time, src, dest > \mid \varepsilon < fid, fdate, time, src, dest > flight \wedge dest = "New\ Delhi" \}$

(b) Get the details about all flights from Chennai to New Delhi

$\sigma_{dest="New\ Delhi" \wedge src="Chennai"}(flight)$

select * from flight where
dest = "New Delhi" AND
SRC = "Chennai"

$\{ t \mid flight(t) \wedge t.dest = "New\ Delhi" \wedge t.src = "Chennai" \}$

$\{ < fid, fdate, time, src, dest > \mid \varepsilon < fid, fdate, time, src, dest > flight \wedge src = "Chennai" \wedge dest = "New\ Delhi" \}$

© Find only the flight number for passengers with pid = 123 for flights to Chennai before 06/11/20

Π_{fid} (σ pid = 123 ∧ dest = "Chennai" ∧ fdate < "06/11/20" (flight))

Π fid(σ pid=123 (booking) ⋈ $\sigma_{dest=}$ "Chennai" ∧ fdate < 06/11/20 (flight))

select fid from ~~flight where~~ flight ⋈ natural join booking where pid = "123" AND Dest = "Chennai" AND fdate < 06/11/20;

$\{ t.fid \mid flight(t) \wedge (\exists b)(Booking(b) \wedge b.pid = "123" \wedge t.dest = "Chennai" \wedge t.fdate < 06/11/20 \}$

2022/6/6

(d) Find the passenger names for passengers who have booking on at least one flight.

Π_{pname} (passenger ⋈ booking)

Select pname from passenger natural join booking ;

$\{ t.pname \mid Passenger(t) \wedge (\exists b)\ booking(b) \wedge b.pid = t.pid \}$

$\{ < pname, pid, pgender, pcity > \mid$

$\exists$ (pid, pgender, pcity, aid, Aid, date)

$\in$ Passenger $< pid, Pname, Pgender, P city > \in$ Pass

$\in < aid, Aid, date > \in$ Booking $\wedge$

pid =

(c) Find the passenger names ~~who donot~~ for those who do not have any booking in any flights.

~~Πpname (~~

Πpname (Π ~~pname~~ pid (passenger) −

Πpid (booking ⋈ passenger))

select pname from passenger where

pid = (select pid from passenger)

minus

{ t.pname | Passenger(t) ∧

group by, order by, having where

→ Subquery :-

A subquery is a query within another SQL query & embedded within the where clause.

Rules

(a) it can be placed within a no of clauses like where, having, from
(b) always enclosed in parantheris & placed on right side of SQL operator
(c) we donot use order by we use group by
(d) if a subquery returns more than one value we have to use multivalued attribute
(e) We can use between operator within the subquery but not with the subquery.

(a) Single Row Subqueries :-
returns 0 or more rows to the outer subquery.

Eg → to display 2nd highest salary

(b) Multirow Subquery
returns more than one value

Eg → to display highest salary department wise & name of employee

2022/01/

12 :- 1:30

who is taking that salary.

~~select dept group~~

select Empname from Emp where salary in (select dept, (max) salary from Emp group by dept);

(c) Nested subquery

a subquery inside another subquery

(d) Correlated Subquery :-

used for row by row processing

Eg) details of employee working on atleast one project.

select * from Emp where Eid exists (select Eid from project where ~~Eid=~~ Emp. Eid = project. Eid);

2022/07/06

Ans

Group By	Having
used to group similar data and can be used with aggregate and gets where, order by	cannot be used with aggregate functions

Having	Where.
cannot be used without group by	cannot be used with aggregate

→ View :-

In some cases it is no disirable for users to see the entire logical model ie the actual relations stored in the database view. So for this purpose we use view that is a virtual table that doesn't really exist in ~~table~~ db.

ⓐ Create View :-

Create view Viewname As
select column1, column2 from
tabname where cond^n;

Indexes → Indexes are special lookup tables that the database engine can used to speed up the data retrieval.

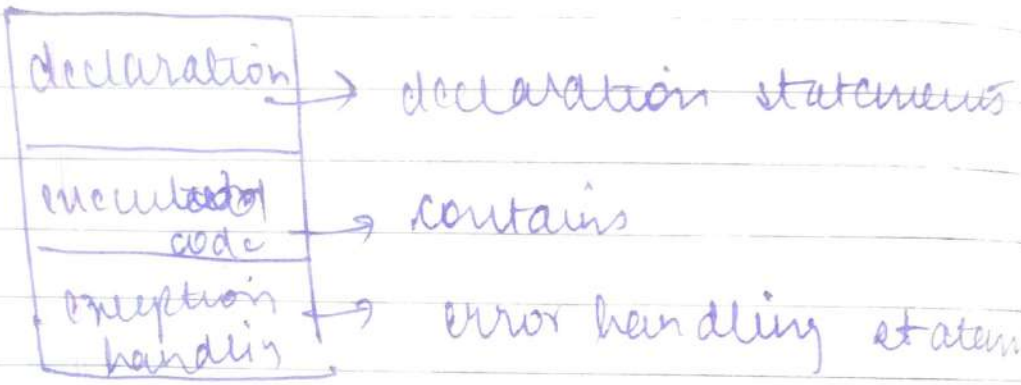
An index is a pointer to the data in the table.

Syntax → create index indexname on
tablename;

20221618

→ PL/SQL :-

It is a block structured language that enables developers to combine the power of SQL with procedural statements



ⓐ Triggers

Triggers are stored programs that are automatically executed or fired when some event occurs

~~Triggers are written to be executed in response to all :-~~

~~ⓐ DML~~

~~DDL~~

(b) Cursor :-

~~Cursor is a pointer do this context~~
Oracle creates a memory space known as context area for processing SQL statements.
A cursor is a pointer to this context area.

(c) Functions :-
It allows us to carry operations that would normally take several queries

(d) Procedure -
are

(e) Functions -
These subprograms return a single value mainly used to compute and return a value

(d) Procedures -
subprograms that donot return a value. Mainly used to perform actions.

Unit - 2

Rdbms	dbms
① stores data in tabular form	① stores raw data in file.
② Normalization is present	② Normalization ✗
③ supports distributed db	✗
④ deals with large data + multiple users.	deals with small data & single user

→ Integrity Constraints -

① Integrity constraints are set of rules that are used to maintain the quality of information

② It ensures that insertion, updation & deletion are performed in such a way that the data integrity is not affected.

(a) Domain Constraints

Domain constraints define valid set of values for an attribute

Eg

ID	Name	Semester	Age
101	X	3	17
102	Y	5	18
103	Z	8	(A)

↓ not allowed

2022/6/6

Eg Create Domain customer-name (
check (value not NULL)

(b) Entity Integrity Constraints

It states that the primary key value cannot be NULL

Eg:

	ID
A	0
B	1
C	

← cannot be null

~~Eg → Create table students (~~

(c) Referential Integrity Constraint

(1) This constraint is applied when a foreign key references to the primary key of a table.

(2) It specifies that all the values taken by the foreign key either must be available in the relation of primary key or be NULL

Eg:

EmpName	Name	Age	DNO. ← foreign key
1	X	1	11
2	Y	2	24
3	Z	3	15
4	P	4	16

Primary Key →

DNO	City
11	Pune
24	Delhi

④ Key Constraints –

Key constraints are used to uniquely identify an entity set.

Eg

ID	Name	Sem	Age
1			
2			
3			
2			

cannot be same

→ Join operations — cross product + ~~some condition~~ selection

Join operation are used to merge tuples from two different Relations

(a) Inner Join — Natural Join, Equi Join, Theta Join

(b) Outer Join — Left outer Join, Right outer Join, Full outer Join

~~(a) Natural Join~~

(a) Inner Join -

(i) Natural Join :-

- denoted by ⋈
- atleast one common attribute
- ~~it is cross product + some condition~~

Eg) Employee

Emp Code	Emp Name
1	A
2	B
3	C
4	D

Salary

Emp Code	Salary
1	100
2	200
4	300

$\Pi_{EmpName, salary}$ (Employee ⋈ salary)

select Emp Name, salary from Employee Natural join Salary;

Output -

Emp Name	Salary
A	100
B	200
D	300

(iii) Equi Join - $A \bowtie A.column1 = B.column2$ (B)

- denoted by =
- No need for column names to be same
- the result can have repeated column names
- We can perform equi join operation on more than 2 tables

~~A, column~~

Eg

EmpNo	Ename	Address
1	A	P
2	B	Q
3	C	R
4	D	S

Employee

Department

DeptNo.	location	Eno
D1	X	2
D2	Y	3
D3	Z	1

Select Ename from ~~cross product~~
Employee, Department
where Employee.EmpNo = Department.Eno
and Employee.address =
Department.location ;

Opt Output :- A

(iii) Theta Join -

- denoted by $\theta \bowtie$
- it combines tuples from different relations provided they satisfy the θ condition

Syntax $S_1 \bowtie_\theta R_1$

Eg)

S1

Sid	name	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

R1

Sid	bid	day	
22	101	10/10/1	96
58	103	10/12	/ 96

$S1 \bowtie_{S1.id < R1.sid} R1$

S1.sid	sname	rating	age	R1.sid	bid	day
22	dustin	7	45.0	58	103	
31	lubber	8	55.0	58	103	

OR $\sigma_{S1.sid < R1.sid}(S1 \bowtie R1)$

2022/6/6 18:57

(b) Outer Join

(i) left Outer Join

(R1 ⟕ R2)

all the ~~relations~~ tuples from left relation R1 are included. The tuples of R1 that donot satisfy the condition are taken NULL

Eg) ~~Cid Name~~

Courses

Cid	Course
100	Database
101	Mechanics
102	Electronics

HOD

Cid	Name
100	Rohan
102	Sara
104	Jiya

Courses ⟕ HOD

Cid	Course	HOD
100	Database	Rohan
102	Electronics	Sarah
100	Database	NULL

select ~~Courses~~. * from Courses.

left join HOD on Courses.cid = HOD.cid;

2022/06/6

(iii) Right Outer Join—

⟖

All tuples from the right relations are included. The tuples of R2 that doesn't satisfy the condition are taken NULL

Eg

Courses

Cid	Courses
100	Database
101	Mechanics
102	Electronics

HOD

Cid	Name
100	Rohan
102	Sara
104	Jiya

Courses ⟖ HOD

Cid	Courses	Name
100	Database	Rohan
102	Electronics	Sara
104	NULL	Jiya

```
select * from Right Outer Join
where Courses.cid =
      HOd.cid;
```

MCQ

(iii) Full Outer Join –

denoted by ⟗

all tuples from both left relation R1 & right relation R2 are included in the relation

Courses

Cid	Course
100	Database
101	Mechanics
102	Electronics

HOD

Cid	Name
100	Ronan
102	Sara
104	Jiya

Courses ⟗ HOD

Cid	Course	HOD
100	Database	Ronan
102	Electronics	Sara
101	Mechanics	NULL
104	NULL	Jiya

Select * from course, Full Outer Join HOD where Courses.cid = HOD.did;

2022/06/6

essential

(i) Check the attribute which doesnot exist in the right hand side.

R(ABCD) — essential attribute = A (is not present on RHS)
A → B
C → D
B → C

$(A)^+ = ABCD$ — AD
$(D)^+ = D$ — BD, CD

(ii) Find the closure of essential attribute if it is not key goto step 3

(iii) Add essential attribute with the other attributes.

$(AD)^+ = ABCD$
$(BD)^+ = BCAD$
$(CD)^+ = CDBA$
} all three are candidate key

candidate key = {AD, BD, CD}

Ques) R(ABCD)
AB → CD
D → A

essential attribute = B

$(AB)^+ = ABCD$
$(BC)^+ = BC$ ✗
$(BD)^+ = BDAC$
} candidate key

candidate key = {AB, BD}

BC → D
BC → A

$(BCD)^+ = BCDA$ } ✗ not candidate key
$(BCA)^+ = ABCD$

Ques) R(ABCDEF)

AB → C
C → D
B → AE

$(AB)^+ = ABCDE$
$(C)^+ = D$
$(B)^+ = A$

essential attribute = B, F

$(BF)^+ = ABCDEF$ BF AECD

candidate key = BF

Ques) R(ABCD)

AB → CD
C → A
D → B

no essential attribute

$A^+ = A$
$B^+ = B$
$C^+ = CA$
$D^+ = DB$
} go for superset

$AB^+ = ABCD$
$AC^+ = ACD$
$CD^+ = CDBA$
$AD^+ = ADBC$
$BC^+ = BCAB$
$BD^+ = BD$

candidate key = {AB, AD, BC}

Ques) R(ABCDE)

AB → CD

D → A

BC → DE

essential attribute = B

$B^+ = B$

$(AB)^+ = ABCDE$ ✓

$(BC)^+ = ABCDEA$ ✓

$(BD)^+ = BDACE$ ✓

$(BE)^+ = BE$

Ques) R(wxyz)

z → w

y → xz

xw → y

essential attribute = x

$w^+ = w$

$x^+ = x$

$y^+ = xzyw$ → super key

$z^+ = wz$

Candidate key = y

$(wx)^+ = wxyz$

$(wy)^+ = wyxz$ → super key

$(wz)^+ = wz$

$(xz)^+ = xzwy$

[wx, xz] candidate key

Normalization → to remove redundant data

→ 2nd Normal form :-

R(ABCD)

AB → D

B → C

(i) find candidate key

AB

(ii) Prime attribute

A B

Non Prime attribute

C D

Condn → (i) Should be in 1NF

(ii) No partial dependency

Ques R(ABC)

B → C

essential attribute = AB

(AB)+ = ABC

① find one candidate key = AB

Prime attribute = AB

non prime attribute = C

R1(AB) R2(BC)

R

A	B	C
a	1	7
b	2	4
a	3	2
c	3	2
d	3	2
e	3	2

R1

A	B
a	1
b	2
a	3
c	3
d	3
e	3

R2

B	C
1	7
2	4
3	2

Ques) R(ABCDE)

$AB \rightarrow C$

$D \rightarrow E$

(i) Find candidate key.

essential attribute = ABD

candidate Key → ABD $(ABD)^+ = ABDCE$

prime = ABD

non prime = C, E

partial dependency

$D \rightarrow E$.

$A \rightarrow C$.

$R_1(ABC)$

$R_2(DE)$

$R_3(ABD)$

Ques) R(ABCDE)

$A \rightarrow B$

$B \rightarrow E$

$C \rightarrow D$

essential attribute = AC

$(AC)^+ = ACBDE$.

candidate key = AC

prime = AC

not prime = BDE

R1(AC)

R2(

partial dependency

$A \to B$

$C \to D$

$R_1(AB)$ $R_2(ABE)$ $R_1(ABE)$ $R_2(AC)$ $R_3(CD)$

Ques) $R(ABCDEFGHIJ)$

$AB \to C$ $R_1(ABC)$

$AD \to GH$ $R_2(ADGHIJ)$

$BD \to EF$ $R_3(BDEF)$

$A \to I$ $R_4(AI)$

$H \to J$

essential att = ABD

prime = ABD not prime = CEFGHIJ.

partial dependency

$A \to I$ $AB \to C$

$H \to J$ $AD \to GH$

$BD \to EF$

$R_1(ABD)$

$R_1(AI)$

$R_2(ABC)$

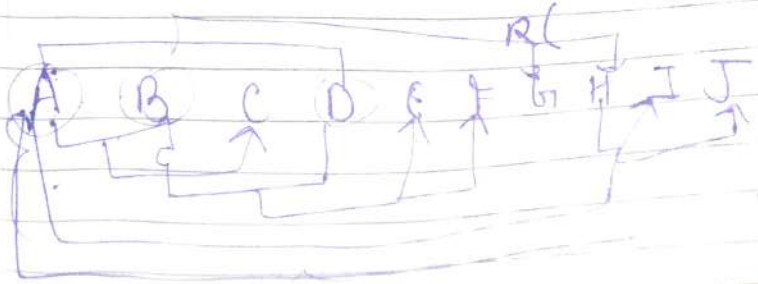
$R_3(ADGHJ)$

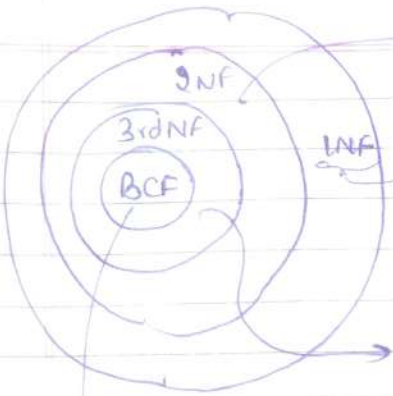
$R_3(BDEF)$

ABD

R(ABD)

R(





Properties
① must be 1NF
② no partial dependency

Properties
by default relation in table is in 1NF (single value must not have multiple values)
it may or may not be 3rd NF
no transitive dependency

1st NF
2nd NF

BCNF → 3rd NF + 2NF + 1NF

2nd →

→ 3rd Normalization form :-

Transitive Dependency :-
from $\alpha \to \beta$ is called transitive if $\alpha, \beta \in$ nonprime

Ques) R(ABC)
$A \to B$
$B \to C$ } transitive relation

essential attribute = A
$(A^+) = ABC$
candidate key = {A}
if {AB} then partial dependency is present
no partial dependency
Thus, it is in 2nd normal form.

prime = A
non prime = BC

$B \rightarrow C$ a B

both production and B are non prime

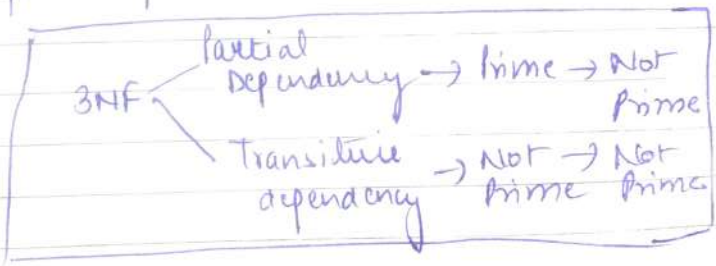
$R_1(AB)$
$R_2(BC)$

Properties

1. It is in 2NF.
2. No transitive dependency

Every dependency from $\alpha \rightarrow \beta$

(i) Either α is super key
(ii) or β is prime attribute



(Ques) R(ABCDE)

$A \to B$

$B \to E$

$C \to D$

essential attribute = A, C

$A^+ = ABE$

$(AC)^+ = ACBED$ ✓ candidate key

(AB, AC, AC)

prime attribute = A, C

Not in 3rd NF

R(ABCDE)

- $R_1(ABE)$ (∵ transitive not 3rd NF)
  - $R_{11}(AB)$ } in 3rd NF
  - $R_{12}(BE)$ }
- $R_2(CD)$
- $R_3(AC)$

NOT

Ques) R(ABCDEFGHIJ)

AB → C

A → DE

B → A

F → GH

~~D → IJ~~ D → IJ

AB → candidate key

prime attribute = AB

R(ABCDEFGHIJ)

- → R_1(ABC)
- → R_2(ADEIJ) → R_{21}(ADE), R_{22}(DIJ)
- → R_3(ABF)
- → R_4(DJ)
- → R_5(BFGH) → R_{51}(BF), R_{52}(FGH)

Ques) R(ABCDE)

AB → C

B → D

D → E

Ans → 3 tables

Ques) R(ABCDEFIJ)

AB → C

AD → GH

BD → EF

A → I

I → J

Ans → 6 tables

→ BCNF :-

(Boyce Codd Normal Form)

1) α must be a supernkey

2) 3NF

determinant

Ques) R(ABC)

$AB \to C$

$C \to B$

$A^+ = A$

$B^+ = B$

$AB^+ = ABC$

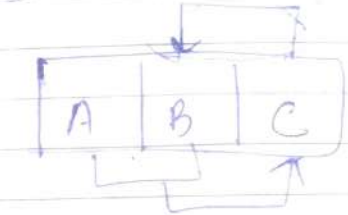
$AC^+ = ABC$

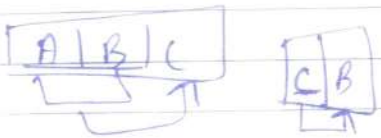
CK = { AB^+, AC^+ }

$AB \to C$ (supernkey)

$C \to B$ (X)

~~R1(ABC)~~





R(~~AB~~

R1 A

R(ABC)

R(ABC)

~~R(AB)~~

R(AC)

R(ABC)

R(A

C is key

R(CB)

R(AC)

R(AB) X

α super key
β prime.

Ques) R(ABCDEFGH)

AB → C ✓ BCNF

A → DE ✗ BCNF ✗ 3NF ✗ 2NF } 1NF

B → F ✗ BCNF

F → GH ✗ BCNF

AB

CK = {AB}

prime = A, B

not prime = C, D, E, F, G, H

Ques) R(ABCDE)

CE → D ✓ BCNF

D → B ✗ BCNF · ✗ 3NF

C → A ✗ 2NF 2NF

EA = CE (NF)

CK = CE

not prime = ABDE

Ques) R(ABCDEF)

AB → C ✗ BCNF ✓ 3NF

DC → AE ✗ BCNF ✗ 3NF

E → F ✗ 2NF } 1NF

CK = {BDA, BDC}

non prime = EF

prime = ABCD

Ques)

R(ABCDEFGH)

AB → C ✗ BCNF ✗ 3NF

A → DE ✗ 2NF } 1NF

B → F

F → GH

CK = ABD

not prime

= {E F G H}

Ques) R(ABCDE)

AB → CD ✓ BCNF

D → A ✗ BCNF ✓ 3NF

BC → DE ✓ BCNF

CK = {AB, BC, BD}

non prime

= EF

3NF.

imp

Ques) R(ABCDEF)

A → BCDEF ✓BCNF

BC → ADEF ✓BCNF

DEF → ABC ✓ BCNF

no essential attribute

A^+ → ABCDEF

BC^+ → ABCDEF

DEF^+ → ABCDEF

Candidate key ⇒ (A, BC, DEF)

Ques) R(ABC)

AB → C ✓BCNF

C → A ✗ BCNF ✓3NF

B

B^+ → B

AB^+ → ABC (3NF)

BC^+ → ABC

(AB^+, BC)

non prime ✗

Anamolies
→ insertion
→ deletion
→ updation

3rd Unit

normalization

1NF

Functional Dependency

Candidate key

2NF

3NF

BCNF

GT Syllabus

→ Lossless Decomposition (Join)

This property guarantees that extra or less tuple generation problem does not occur after decomposition

If a relation R is decomposed into two relation R_1 & R_2 then it will be lossless if and only if

(i) $att(R_1) \cup att(R_2) = att(R)$

(ii) $att(R_1) \cap att(R_2) \neq \phi$

(iii) $att(R_1) \cap att(R_2) \rightarrow att(R_1)$

OR

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

Common attribute must be a candidate key

If these properties are not followed ⇒ lossy

Ques

R

A	B	C	D
1	a	p	x
2	b	q	y

$R_1(AB)$

$R_2(CD)$ → cross product

A	B
1	a
2	b

C	D
p	x
q	y

A	B	C	D
1	a	p	x
1	a	q	y
2	b	p	x
2	b	q	y

$R_1 \bowtie R_2$

A	B	C
1	a	p
2	b	q
3	a	r

$R_1(AB)$

A	B
1	a
2	b
3	a

$R_2(BC)$

B	C
a	p
b	q
a	r

A	B	C
1	a	p
1	a	r
2	b	q
3	a	p
3	b	r

lossy

R

A	B	C	D	E
a	122	1	p	w
b	234	2	q	x
a	56	1	r	y
c	347	3	s	z

lossy

(i) $R_1(AB)\ R_2(CD)$ — 1st ✗

(ii) $R_1(ABC), R_2(DE)$ — 2nd ✗

(iii) $R_1(ABC)\ R_2(CDE)$ — 3rd ✗

(iv) $R_1(ABCD)$ $R_2(ACDE)$

lossy by (iii) property

(iv) $R(ABCD)\ R_2(ACDE)$

⇓

lossless

Date
Page

Ques) $R(V, W, X, Y, Z)$

$Z \to Y$
$Y \to Z$
$X \to YV$
$VW \to X$

#Note (*)

$YV \to X$
$Y \to X$
$V \to X$

$X \to YV$
$X \to Y$
$X \to V$

(i) $R_1(VW)$ $R_2(YZ)$ lossy (1st property)

(ii) $R_1(VWX)$ $R_2(YZ)$ lossy (and)

(iii) $R_1(VW)$ $R_2(WXYZ)$

essential attribute = W ~~W → candidate key~~ common attribute = W

$W^+ = W$ so W is not candidate key

not lossless and not lossy

(iv) $R_1(VWX)$ $R_2(XYZ)$

~~X → candidate key~~

~~essential attri~~ common attribute = X

$X \to$ ~~not candidate key~~

not lossless

not lossy

$X \to YV$

$X \to Y$

$X \to V$

bcoz of (*) $R_2(XYZ)$ → X is not candidate key for R_1 for $R_1(VWX)$

So, X is candidate key

Hence lossless

This step is optional and whether the data is lossy or not depend upon it.

Dependency Preservation

If a table R having functional dependency set F_1 is decomposed into two tables R_1 & R_2 having functional dependency set F_1 & F_2 then

$F_1 \subseteq F$

$F_2 \subseteq F$

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

Ques) R(A B C)

$A \to B$

$B \to C$

$C \to A$

R(AB) | R(BC)

$A \to B$ | $B \to C$

$B \to A$ | $C \to B$

$C \to A$ ($\because$ $B \to A$, $C \to B$)

Hence dependency is preserved

Ques) R(ABCD)

AB → CD

D → A

(D) → unique key (loss less decomposition)

$R_1(AD)$

A → D ✗

D → A ✓

$R_2(BCD)$

B → ✗

C → ✗

D → A ✗ A is not part of R_2

BC → ✗

BD → BD → A C

CD → ✗

X

↓

not part of R_2

$A^+ = A$

BD → BD

BD → A

BD → C ✓ → use parent

for R_1

D → A

for R_2

BD → C

$(BD)^+$ → BDC

Not equal → use avoid

$(AB)^+ = ABCD$

No dependency preservation

Either the dependency or its closure must match.

Ques) R(ABCDEFGHIJ)

AB→C ✓ BCNF

A→DE ✗ BCNF ✗ 3NF ✗ 2NF (1NF)

B→F ✗ BCNF

D→IJ

F→GH

AB^+ = ABCDEFGHIJ.

prime = AB

non prime = CDEFGHIJ.

Convert to 2NF from 1NF

① candidate key ~~$R_1(AB)$~~ $R_1(ABC)$

~~$R_2($~~ $R_2(ADEIJ)$

② partial key $R_3(BFGH)$

$R_4(GH)$

B→F

F→GH

$B^+ \rightarrow BFGH$

Convert to 3NF

① ~~Non Prime~~ not candidate key → non prime

$A \rightarrow DE$

$R_1(ABC)$

$R_2(ADEIJ)$

FGH

$R_{21}(ADE)$

R

ADEIJ

A→DE

D→IJ

(A)

$A^+ \rightarrow$ ADEIJ

A

DEIJ.

$R_1(AB)$

$R_2(ADEIJ)$

$R_{21}(ADE)$

$R_{22}(DIJ)$

$R_3(BFGH)$

$R_{31}(BF)$

$R_{32}(FGH)$

2022/6/6 1

→ **4NF :-**

- no multivalued dependency
- BCNF

→ **5NF :-**

- 4NF
- no joint dependency

→ **Multivalued Dependency :-**

$A \twoheadrightarrow B$

(1) for single value A there are more than one value B

A	B
A_1	B_1
A_1	B_2

(2) A table should have atleast 3 column to have multivalued dependency.

(3) Column B and C should be independent

A	B	C
A_1	B_1	C_1
A_2	B_2	C_2

2022/6/6

Ques)

Sid	Course	Hobby
1	Science	~~Sanjay~~ Singing
1	Math	Reading
2	Chemistry	Singing
2	Physics	Reading

Multivalued attribute

convert into 2 tables

Course table

Sid	Course
1	Science
1	Math
2	Chemistry
2	Physics

Hobby table

Sid	Hobby

When function and multivalued dependency occur simultaneously we convert it into 3 tables.

Eg

Sid	Course	Hobby	address

Sid	Course

Sid	Hobby

Sid	Address

MVD → Multivalue dependency
JD → Join dependency

PJNF

→ 5NF :- (Project Joint Normal Form)

(i) 4NF

(ii) NO join dependency
↳ it can't be further non-loss decomposition

Join Dependency :- eg $R_1, R_2, R_3 \dots R_n$

Let 'R' be a relation schema & R is said to satisfy the join dependency iff

$$\Pi_{R1}(R) \bowtie \Pi_{R2}(R) \bowtie \dots \Pi_{R_n}(R) = R$$

eg)

Agent	Company	Product
Aman	C_1	PD
Aman	C_1	Mic
Aman	C_2	Speaker
Mohan	C_1	Speaker

R_1

Agent	Company
Aman	C_1
Aman	C_2
Mohan	C_1

R_2

Agent	Product
Aman	PD
Aman	Mic
Aman	Speaker
Mohan	Speaker

R_3

Company	Product
C_1	PD
C_1	mic
C_2	speaker
C_1	speaker

$\Pi_R R_1 \bowtie R_2$ → in this composite primary key

Agent	Company	Product
Aman	G	PD
Aman	G	mic
Aman	G	Speaker
Amo	C_2	PD
Aman	C_2	Mic
Aman	C_2	Speaker
Mohan	G	Speaker

it is in 5th NF
Since it has lossy decomposition

Ques

Name	Skill	Job
Aman	DBA	J1
Mohan	Tester	J2
Rohan	Programe	J3
Sohan	Analyst	J4

5NF? Not in 5NF

$\Pi_R R_1 \bowtie R_2$

Name	Skill
Aman	
Aman	
Aman	
Aman	

Name	Job

2022/6/8

Unit - 3

① Functional dependencies

determinant → $\alpha \rightarrow \beta$ ← dependent

$\alpha \subseteq R \quad \beta \subseteq R$

eg

R	α	β
t_1	α	β
t_2	α	β

if $t_1[\alpha] = t_2[\alpha]$
↳ $t_1[\beta] = t_2[\beta]$

$\alpha \rightarrow \beta$

Trivial

$AB \rightarrow A$

$\beta \subseteq \alpha$

Non trivial

$\beta \not\subseteq \alpha$

$AB \rightarrow ABC$ (some new info)

Ques) Which of the following dependency holds?

A	B	C	D	E
a	1	3	4	5
2	a	3	4	5
a	2	3	6	5
a	2	3	6	6

a) $A \rightarrow BC$

b) $DE \rightarrow C$

x c) $C \rightarrow DE$ (for 3 different values)

d) $BC \rightarrow A$

for not valid => atleast one value of α different value of β same

or not holds.

$X \to YZ$ ✓
$X \to Y$
$Y \to Z$

$YZ \to X$ ✗
$Y \to X$
$Y \to Z$

→ Attribute closure / Closure on attribute set / closure of attribute set :-

Attribute closure of an attribute set 'A' can be defined as a set of attributes which can be functionally determined from it. They are denoted by 'F^+'.

Eg R(ABC)

$A \to B$

$B \to C$

$A \to BC$

$(A^+) = ABC$

→ Equivalence of functional dependencies :-

Eg R(ACDEH)

F: $A \to C$
$AC \to D$
$E \to AD$
$E \to H$

G_H: $A \to CD$
$E \to AH$

and values from here

their closure

$A^+ = ACD$
$AC^+ = ACD$
$E^+ = AHECD$

all the words can be compared with G

So, $F \subseteq G$

$A^+ = ACD$

$E^+ = EHADC$

$G \subseteq F$

Thus $\boxed{F = G}$

Eg) $R(PQRS)$

X: $P \to Q$, $Q \to R$ ✗, $R \to S$

Y: $P \to QR$ ✓, $R \to S$ ✓

$P^+ = PQRS$; $P^+ \to PQRS$

$Q^+ = Q$; $R^+ \to RS$

$R^+ = RS$; $Y \subseteq X$

$X \not\subseteq Y$

$X \neq Y$

Thus $\boxed{Y \subset X}$

→ efficient super key

→ Super Key & Candidate Key :-

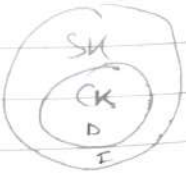
$(SK)^+ = R$

$R(ABCD)$

$ABC \to D$

$AB \to CD$

$A \to BCD$



$(ABC)^+ = ABCD$

$(AB)^+ = ABCD$ } all three are super key

$(A^+) = ABCD$

$A^+ \to$ candidate key

Primary key → any one candidate becomes PK.

→ Functional Dependencies

① Functional dependency in DBMS is a relation ~~between~~ between attributes of a table dependent upon each other.

② It helps in preventing data redundancy and helps us to get to know about bad designs

$A \rightarrow B$

B → B functionally dependent on A

A → determinant set

B → determinant attribute

Eg

	α	β	γ
t_1	α_1	β_1	γ_1
t_2	α_2	β_4	γ_2
t_3	α_1	β_1	γ_1
t_4	α_2	β_5	γ_2

if $t_1[\alpha] = t_3[\alpha]$

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

$\alpha \rightarrow \beta$

$\alpha \not\rightarrow \beta\gamma$

→ Types :-

(a) Trivial functional dependency

① A → B has trivial functional dependency if B ⊆ A, ie B is a subset of A

② following dependencies are also trivial A → A, B → B

Eg)

Eid	Ename
1	xyz
2	abc
3	pqr

① {Eid, Ename} → Eid is a trivial dependency as

② Eid is a subset of {Eid, Ename}

③ Also,
Eid → Eid & Ename → Ename are trivial dependencies

delete
from tablename
[where condn];

foreign key (Pid)
references
tablename (Pid)

(b) Non-trivial Dependency

(1) $A \to B$ has non trivial dependency if $B \not\subseteq A$, ie B not a subset of A

(2) if A intersection B ($A \cap B = \phi$) then $A \to B$ is called complete non trivial dependency.

Eg) $ID \to Name$
$Name \to DOB$
$A \cap B = \phi$
↳ complete non trivial dependency

(c) Partial Dependency

(1) It occurs when a non prime attribute is functionally dependent on a part of candidate key.

(2) 2NF removes partial dependency

Eg) Student id | Project id | student name | Project name

Student name can be determined by student id so partially dependent

Project name can be determined by project id so PD

④ Transitive Dependency

① A functional dependency is said to be transitive dependency if in $\alpha \to \beta$, $\alpha\beta$ are & non prime attribute.

② Helps us in normalizing the database in 3NF form

Eg $R(ABC)$

$A \to B$

$B \to C$

$A^+ = ABC$

prime attribute = A

not prime attribute = BC

$B \to C$ has transitive dependency

Partial dependency

Prime Attribute → Not Prime attribute

Transitive dependency

Not Prime → Not Prime

→ Inference Rules

① Armstrong's axioms are known as inference rules

② They can be applied to a set of functional dependency to derive another functional dependency

Primary Rules

ⓐ IR1 (Reflexive Rule)

If Y is a subset of X then X determines Y

$Y \subseteq X$

$X \rightarrow Y$

RARUDP

RATUDP

ⓑ IR2 Argumentation Rule

Also called partial dependency

$X \rightarrow Y$

$XZ \rightarrow YZ$

Eg) R(ABCD)

$A \rightarrow B$

$AC \rightarrow BC$

ⓒ IR3 (Transitive Rule)

$X \rightarrow Y$

$Y \rightarrow Z$

$X \rightarrow Z$

Secondary Rules → can be prooved using primary rules

(d) IR4 (Union)

$X \to Y$ &
$X \to Z$
then
$X \to YZ$

Proof

$X \to Y$ } (Given) — (1)
$X \to Z$ — (2)

Using IR_2 with X and $X \cdot X = X$ on (1)

$X \cdot X \to XY$

$X \to XY$ — (3)

using IR_2 with Y on (2)

$XY \to YZ$ — (4)

using IR_3 on (3) & (4)

$\boxed{X \to YZ}$

(e) Decomposition Rule (IR_5)

$X \to YZ$
then
$X \to Y$
$X \to Z$

Proof $X \to YZ$ (given) — (1)

IR_1 on (1)

$YZ \to Y$

IR_3 on (1) & (2)

$X \to Y$

(f) IR6 (Pseudo Transitive Rule)

$X \to Y$
$WY \to Z$
then
$WX \to Z$

Proof → $X \to Y$ — (1)

$WY \to Z$ — (2)

IR_2 using W on (1)

$WX \to WY$ — (3)

IR_3 on (2) & (3)

$\boxed{WX \to Z}$

(a) Super key

The closure of a superkey is the entire relation schemma

$$(SK)^+ = R$$

(b) Candidate key

If there exists no subset of an attribute whose ~~relation~~ closure contains all the attributes of the relation then it is called candidate key.

Eg R(ABCD)

$AB \rightarrow D$

$AB \rightarrow CD$

$A \rightarrow BCD$

$(ABC)^+ = ABCD$

$(AB)^+ = ABCD$ } super

$A^+ = ABCD$

candidate key

2022/6/6

→ Equivalance of two set of functional dependencies :-

① F covers G ($F \geq G$)
② G covers F ($G \geq F$)
③ Both F & G cover each other ($F = G$)

→ Type of Normal forms :-

① 1NF :-

A relation is in 1NF if it contains an atomic value.

② 2NF :-

① should be 1NF
② no partial dependency

③ 3NF

① must be 2NF
② no transitive dependency

OR

~~for $\alpha \to \beta$~~ for every dependency $\alpha \to \beta$

either
1) α is a superkey
2) or β is a prime attribute

2022/6/6

④ BCNF

Boyce Codd Normal form

1) α is a super key

2) must be in 3NF.

⑤ 4NF

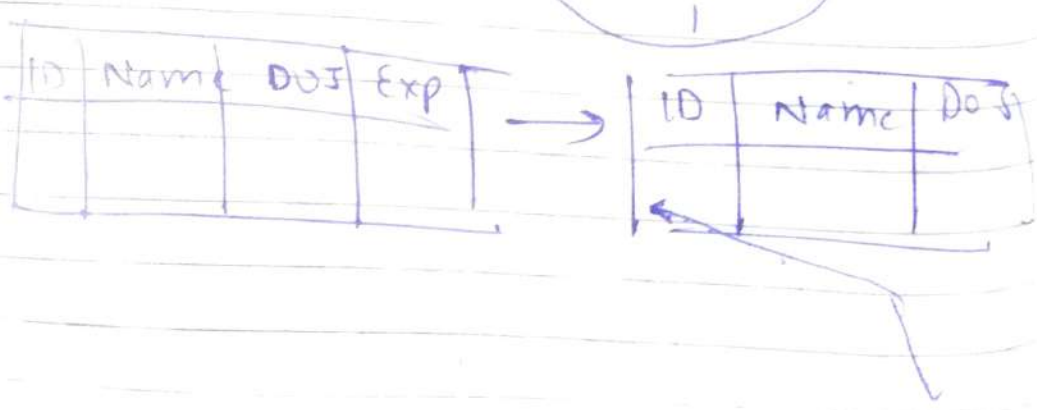
1) BCNF

2) no multivalued dependency

⑥ 5NF

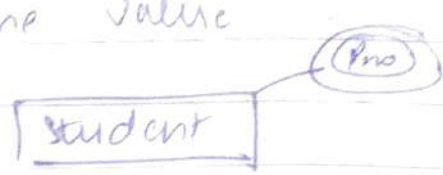
1) 4NF

2) no join dependency & joining should be lossless.



(a) Multivalued Attribute → They may contain more than one value

ID	Name	Age	Phone
1	A	21	1234
2	B	31	5678
3	C	41	9101112



↓ 1NF

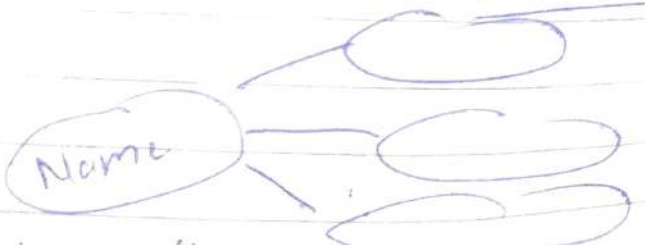
ID	Name	Age	Phone
1	A	21	1231
2	B	31	5678
3	C	41	9101112
3	C	41	13141516

(b) Composite Attribute :-
made up of more than one simple attribute Possible to further break them down.

ID	Name	Address

1NF →

ID	Name	City



(c) Derived attribute

Derived attributes do not exist in database they are derived from some other attribute

In 1NF, 2NF, 3NF we decompose the relation to minimize the dependency but some times are not sufficient. So we add additional properties

→ Types of Decomposition :-

Decomposition in DBMS removes redundancy, anamolies and inconsistencies by dividing the database into multiple tables

1. Lossless Join Decomposition
2. Lossy Join Decomposition

→ Relational Algebra	Relational Calculus
1. Procedural language	declarative language
2. It means how to obtain result	It means what result we have to obtain
3. Independent of domain	can be dependent on domain
4. Order is specified in which operation is to be performed	4. Order is not specified.
5. nearer to programming language	5. not nearer to programming language.

20/2/16

→ Safe Expression :-

Whenever we use universal quantifiers, existential quantifiers or negation of predicate in calculus we must make sure that the resulting expression makes sense.

A safe expression in relational calculus is one that is guaranteed to yield a finite no of tuples as result otherwise the expression is called unsafe expression.

→ SQL

structured query language
defines structure of data

DDL

① Data Definition language
used to define database
structure or pattern
Create → create objects in DB
Alter → alter
Drop → Drop
Truncate →
Rename → ren
Comment →

C A D T R C → stru

② Data Manipulation language
(DML)
used for accessing &
manipulating data

S elect → retrieve data
I nsert → insert val
D elete →
U pdate →
M erge →
E xplain Plan → parameter for explain da
L ock Table
C all ↳ controls concurrency

↳ use to all SQL

2022/04/6

③ DCL

Data Control Language

Grant → used to give access privileges to db

Revoke

↳ to take back permission from users

④ TCL

used to run changes made by DML

(a) Commit → used to save transactions on db

(b) Rollback → restore the db to original since last commit

→ (1) searching, sorting & creating indices is faster
(2) anamolies reduced
(3) reduced redundancy
(4) reduces data dependency
(5) small/nog null values

Normalization

It is the process in which decompose our relationship in more than one relation remove anamolies from the db.

Types of Anomolies –

Insertion Anomaly –

We ~~want to~~ insert some information but we cannot insert it due to some constraints.

Deletion Anomaly –

it occurs when we delete some existing information but it causes deletion of any other undesirable information

Updation Anomaly –

it occurs when we try updating the data and it causes inconsistency of data.

2022/8/5

ACID properties

Unit-4

Transaction System

Serial scheduling → consistency

Non serialized → after completion of one process another is started

Conflict

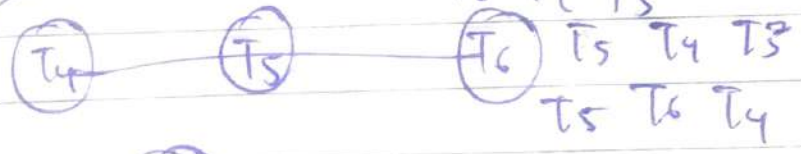
R-R ✗ } non conflicting

R-W ✓

W-R ✓ } conflicting

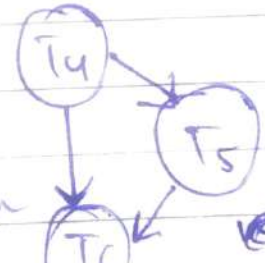
W-W ✓

Rules) Conflict serializability method



$T_4\ T_5\ T_6$
$T_4\ T_6\ T_5$
$T_6\ T_5\ T_4$
$T_6\ T_4\ T_5$
$T_5\ T_4\ T_3$
$T_5\ T_6\ T_4$

n! ⇓ 3! ways in which the transaction can be completed



no cycle then $T_4 \rightarrow T_5 \rightarrow T_6$

conflicted serializable

Precedence Graph

Serializable - consistent → after no change in the data.

A → B

400 300

imp → after performing some operations non serializable becomes serializable

Conflict serializable Schedule:-

after swapping non conflicting operations (R-R) only it can transform into serial schedule

Conflicting Operations

a) both belong to seperate transaction
b) have same data item
c) they contain atleast one write operation

→ Conflict Equivalent:-
One can be transformed to another by swapping non conflicting operations

~~Conflict only if~~

Note
if conflict serializable schedule then it is serial schedule

if not conflict serializable then we cannot tell whether it is serial or not
so for this case we use view serializability method.

→ View Serializability

if schedule is conflict serializable then it will be view serializable.

A schedule will be view serializable if it is equivalent to a serial schedule

Blind write

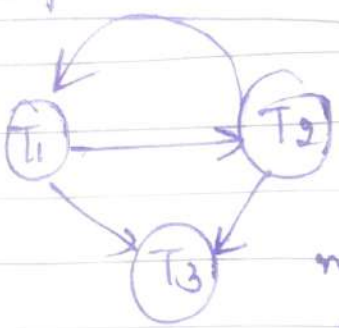
without reading writing.

(a) initial read
(b) update read write
(c) final write

S1

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

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



loop so non conflict serializable schedule

Cannot tell whether

Serial

Non Serial

so we use view serial

~~1) Initial Update~~

- 1) initial read
- 2) update ~~read~~ → w
- 3) final write

} using these three ~~write~~ conditions

by view serializability this is ~~serializable~~ serial.

Transaction Pro

Transaction

Transaction properties

life cycle of Transaction
- active
- committed
- partially committed
- failure
- terminated
- failure

Schedules

→ Recoverability of Schedule

irrecoverable is recoverable with cascade rollback.

Cascade less recoverable schedule

Recovery System

① Deferred
no undo/redo (rollback + redo operations)

② Immediate
if before commit there is failure then old value from log record

Undo/Redo Operation :

After committing if failure occurs → redo
Before committing failure occurs → abort

Log base recovery
keeping information of transaction occuring

Deferred

① <T_i, item, new value> till the time value is not committed it will not go in db

② if before committing failure ⇒ redo
if before committing failure then it

Immediate

① data write immediately in db

② if failure after commit ⇒ redo (take latest value)
if before commit ⇒ starting value as before (undo operation)

→ ~~Checkpoints~~ :-

Concurrency

waiting time ↓ resource utilization ↑
efficiency ↑
drawback → deadlock.
inconsistent data
must be serial. (conflict serializability)

	S	X
S	T	F
X	F	F

Lock Based Protocols

→ Two Phase Locking Protocol :-

Set of rules followed by all transactions while requesting & releasing locks. Locking protocols restrict the set of possible ⓐ schedules
ⓐ Growing Phase
ⓑ Shrinking Phase

→ ~~Lock Con~~ → Lock Conversions

a requirement of the basic two phase

→ Dissuss handling & Distributed DB

Unit-4

→ Transaction
→ lifecycle — Partially committed → write saved in RAM
fully committed → saved in database
→ properties

→ after commit no more changes

→ Serial / non Serial / Serializability
↓
Concurrent execution equal → serial

- irrecoverable schedule
(log based — Deferred / immediate)
Numericals
- Checkpoint
- Deadlock handling & 
Instrument to
Condition
Prevention
Transaction
Deadlock prevention
Scheme → wait bound, Time bound

Deadlock Detection

Recovery — total Rollback

partial Rollback

Concurrency Control

Distributed Systems-

→ Problems occurs during concurrency :-

1) Dirty Read Problem :- (W-R Conflict)

occurs when one transaction updates an item in db but later fails and before the data is rolled back it is accessed by another transaction

T1	T2
R(A)	
W(A)	
↓	
Retrive	R(A) – Dirty Read
Commit	commit

2) Unrepeatable Read Problem (W-R)

occurs when two different values are read for the same db item ambiguity

T1	T2
to R(A)	
	R(A)
	↑ T2 here confused for the value to & t1
t1 W(A)	
	R(A)

3) Phantom Read Problem :-

occurs when a structure is deleted

T1	T2
R(A)	
	R(A)
Delete (A)	
	R(A)
	↑ Variable doesnot exist

occurs when a transaction reads a value but when it tries to read the same value an error occurs coz that value doesnot exist anymore it was deleted

Unit-5 → Time ——

(write 4) **Lost update Problem** it occurs when two different db transaction perform read and write operation on the same database in an interleaved manner making the db inconsistent

T1	T2
R(A)	
A = A + W(A)	
	A = A + 10
	W(A) to commit

commit

Because T1 updated value is lost so it is called lost update problem.

→ **Blind write :-**

Before reading write operation is performed

→

2022/04/0

TS → Timestamp
RTS → Read "
WTS → Write "

Timestamp Based Protocol :-

(a) Unique value assigned to every transaction

(b) Tells the order (when transaction enters the system)

(c) Read - TS(RTS) = Last (Latest) transaction which performed. (read successfully)

(d) Write - TS(WTS) = Last (Latest) transaction which is performed

Rules -

(i) Transaction T_i issues a Read(A) operation

(a) if $WTS(A) > TS(T_i)$ Rollback T_i

(b) Otherwise execute R(A) operation set $RTS(A) = MAX\{RTS(A), TS(T_i)\}$

(ii) Transaction T_i issues a write operation.

(a) if $RTS(A) > TS(T_i)$ then rollback T_i

(b) if $WTS(A) > TS(T_i)$ then rollback T_i

(c) Otherwise execute write(A) operation set $WTS(A) = TS(T_i)$

2022

Ques

100	200	300
T1	T2	T3
R(A)		
	R(B)	
		R(B)
W(C)		
R(C)	W(B)	W(A)

	A	B 300	C
RTS	~~0~~ 100	~~0~~ 300	~~0~~ 100
WTS	~~0~~	~~0~~	~~0~~ 100

for A (1) (a) $0 > TS(T_i) \Rightarrow 0 > 100$ X

(b) $RTS(A) = \max\{ RTS(A), TS(T_i) \}$

$RTS(A) = \max\{0, 100\}$

$= 100$

for B in (T_i) (1) (b) $RTS(B) = 200$

for B in T_3 (1) (1) $0 > 300$ false

(2) in. $RTS(B) = \max\{200, 300\}$

$= 300$

for W(C) in T_1 $\Rightarrow$ (2) (a) $RTS(C) > TS(T_i)$ $0 > 100$ X

(b) $WTS(C) > TS(T_1) \Rightarrow 0 > 100$ X

(c) set $WTS(A) = 100$

20222626

for R(C) in T_1 ⇒ ① 100 > 100 false

② see RTS(C) = max(0, 100)
= 100

for W(B) in T_2 ⇒ ② RTS(B) > TS(T_2)
300 > 200 ✗ ✓ true

~~② ⑥ WTS(B) > TS(T_i)~~

rollback

Rollback occurs at W(B) in T_2

(Late Smashers)

→ Thomas Rule

Problem occurs in multicourse transaction step remove it.

Database
Data

Serializability

Date

→ Multiversion Concurrency Control

Multiple Granularity → to easily get the data

~~conflict~~ ~~serial~~

Extension

ER Diagrams

Unit-4

Transactions

Schedule → It is the chronological sequence of multiple transactions.

- Serial (consistent)
  - disadv → waiting time
    throughput ↓
    no of transactions executed / unit time
  - Eg → ATM
- Parallel
  - multiple transactions at same time
  - Eg → internet banking

Problems in parallel schedule

Read - Write Conflict or unrepeatable read

- R-R → No conflict
- W-R, R-W, W-W → conflicting

Irrecoverable Schedule

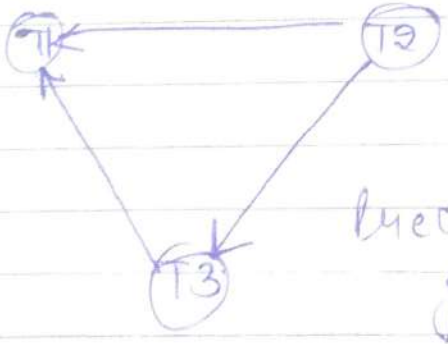
Conflict Serializability

T1	T2	T3
R(x)		
		R(y)
		R(x)
	R(y)	
	R(z)	
		W(y)
	W(z)	
R(z)		
W(x)		
W(z)		

Check conflict pairs in other transactions and draw edges

Conflict pairs

R - W
W - R
W - W



Precedence graph

Check for loop or cycle

no loop in this cycle

Conflict serializable Schedule

Serial exists

Guaranteed that it is A Serializable and Consistent

① Check indegree = 0 (no edge is coming)

② T2

③ T1

T3

remove 3

T1

T1 → T2 → T3
T1 → T3 → T2
T2 → T1 → T3
T2 → T3 → T1
T3 → T1 → T2
T3 → T2 → T1

6 possibilities

T2 → T3 → T1

This is the sequence.

→ Conflict Equivalent

R(A)	R(A)	} non conflict pairs
R(A)	W(A)	} conflict pairs
W(A)	R(A)	
W(A)	W(A)	

R(B)	R(A)	} non conflict pairs
W(B)	R(A)	
R(B)	W(A)	
W(A)	W(B)	

S

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	

S'

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)

Check adjacent non conflict pairs if conflict ⇒ no swap, if not conflict swap

S

T_1	T_2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)

⇒

S

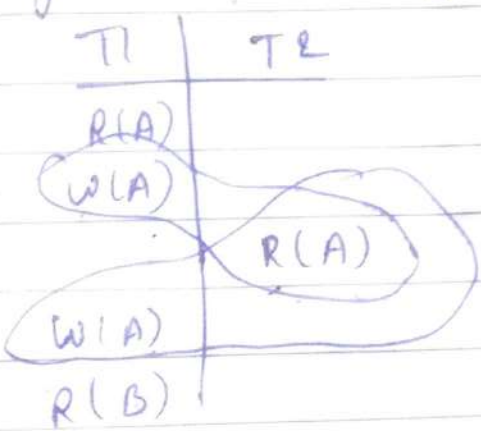
T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)

now S & S' look same

$S \equiv S'$

Note

if conflict no change in positions



if conflict equivalent exist → Serializable

→ Serializability –

S

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

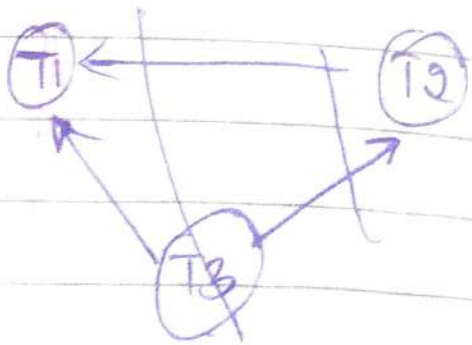
T1 → T2

Serializability — Conflict / View

View Serializability :–

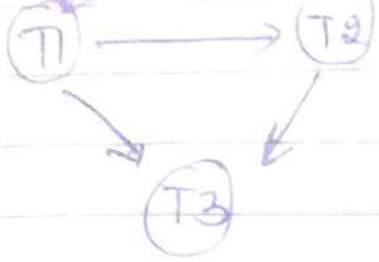
S

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



No loop / Cycle
⇓
Conflict serializable
⇓
Serializable
⇓
Consistent

loop
⇓
Not Conflict serializable
⇓
Cannot tell serializable or not
⇓
solve using view serializability



not conflict serializable

no loop / cycle → conflict serializable
⇓
serial
⇓
consistent

~~# if loop ⇒ non conflict serial~~

if no loop then we cannot tell serializable or not so then we use view serializability

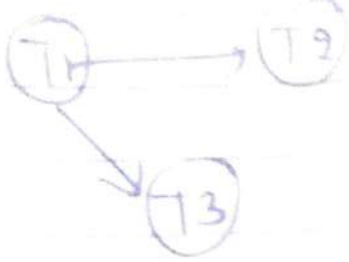
Eg)

A=100	T1	T2	T3
	100 R(A)		
	60 W(A)		
		20 W(A)	
			W(A) 0

	T1	T2	T3
	100 R(A)		A=A-40
		W(A)	
	A-40 W(A) 60		
			A-20 W(A) 0

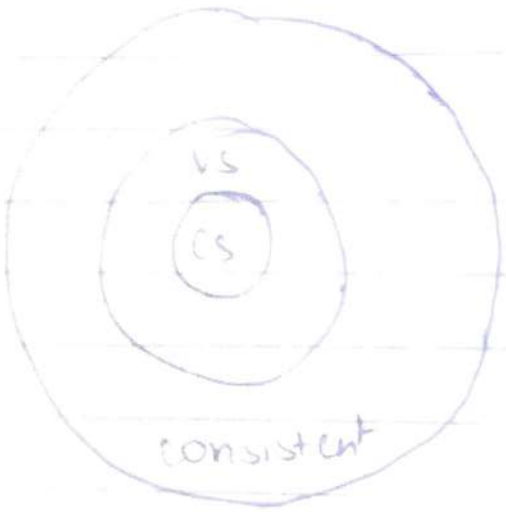
Blind write

Both are equivalent
but don't look equivalent
this is view serializability



Serializable.

If loop in conflict serializable then we use view serializability



Inconsistent

So basically,

conflict serializable

Y ↙

N ↘

View Serializability

Blind write.

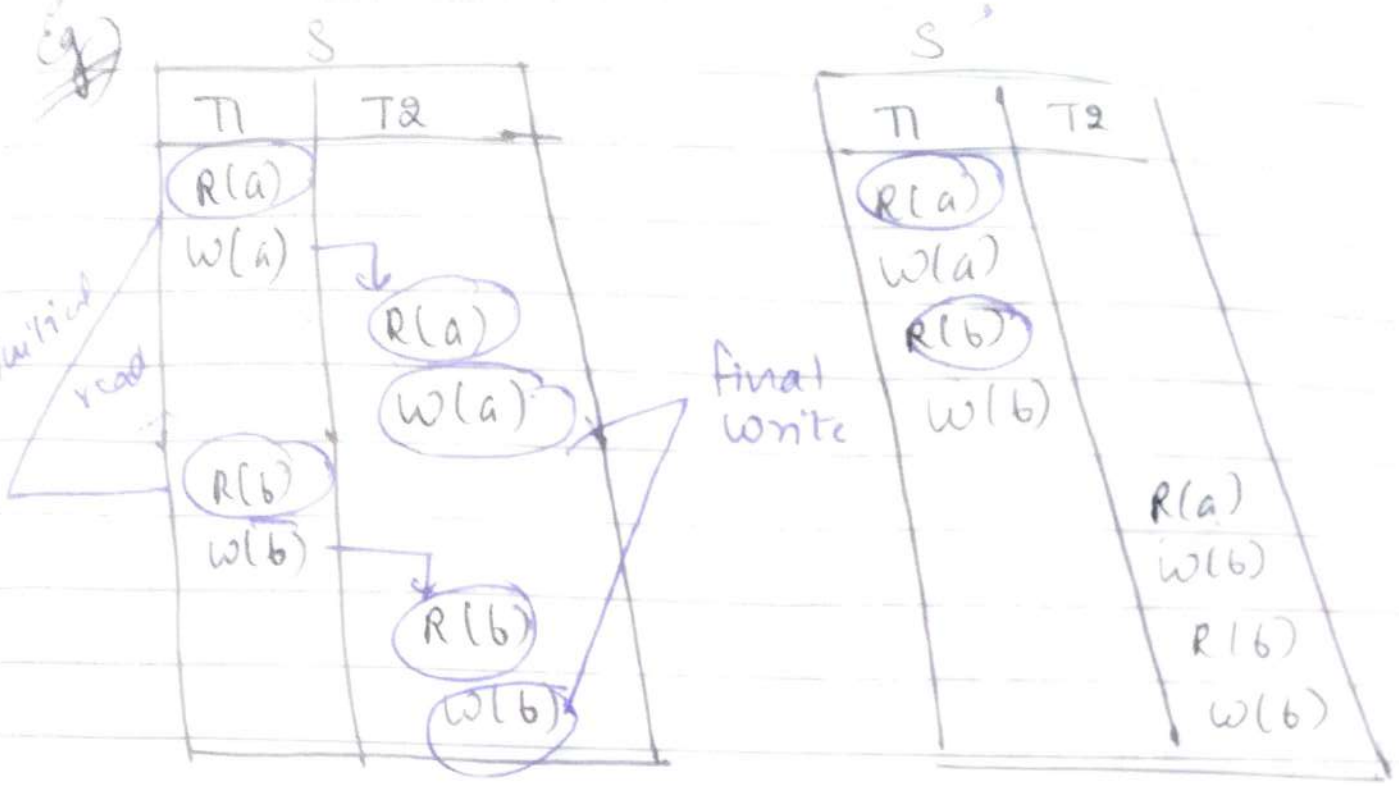
Yes ↙

N ↘

Check View Serializability

VSX

(g)



~~① initial~~

g

① initial read is done by t1 in then in other table also initial read is done by T_1

② final write → if final write is done by T2 in S then in S' also T_2 will do the final write

③ Update read, intermediate read

same

T2 read T1 value (R(a))

T2 reads T value (R(b))

then view equivalent

serializable.

Blind write → if for any transaction without reading we are performing write

Ques)

S

T1	T2	T3
R(a)		
	W(a)	
W(a)		
		W(a)

S1

T1	T2	T3
R(a)		
W(a)		
	W(a)	
		W(a)

T1 T2 - T3

loop so not conflict serializable

Now Check for view serializable.

$n! = 3! = 6$

S1 T1 T2 T3
S2 T1 T3 T2
S3 T2 T1 T3
S4 T2 T3 T1
S5 T3 T2 T1
S6 T3 T1 T2

Disadv: n! cases and have to check for each case and it is conflict.

For ~~T1 T2 T3~~ S1 -

S

T1	T2	T3
R(a)		
	W(a)	
W(a)		
		W(a)

S1

T1	T2	T3
R(a)		
	W(a)	
W(a)		
		W(a)

initial read :- ✓ update read :- ✓
final write :- ✓

S1 and S1 are view serializable so no need to check for S2, S3, S4, S5, &

→ Recoverability of Schedule :-

ⓐ Recoverable Vs Irrecoverable

T2 committed already before failure

T2 committed after failure

ⓑ Cascading Schedule

due to occurence of one event multiple events are occuring

	T1	T2	T3	T4
A=50 A-50 =50	R(A)			
A=50	W(A)			
		R(A) 50		
			R(A) 50	
				R(A) 50

Rollback * failure

reading before commit

after rollback A become 50 but T2 T3 T4 have A value as 50 but 50 in reality doesn't exist. This is called Dirty read problem. Thus we make all these abord

by rolling them back.

Disadv → performance CPU utilization

④ Cascadeless Schedule :-

Eg

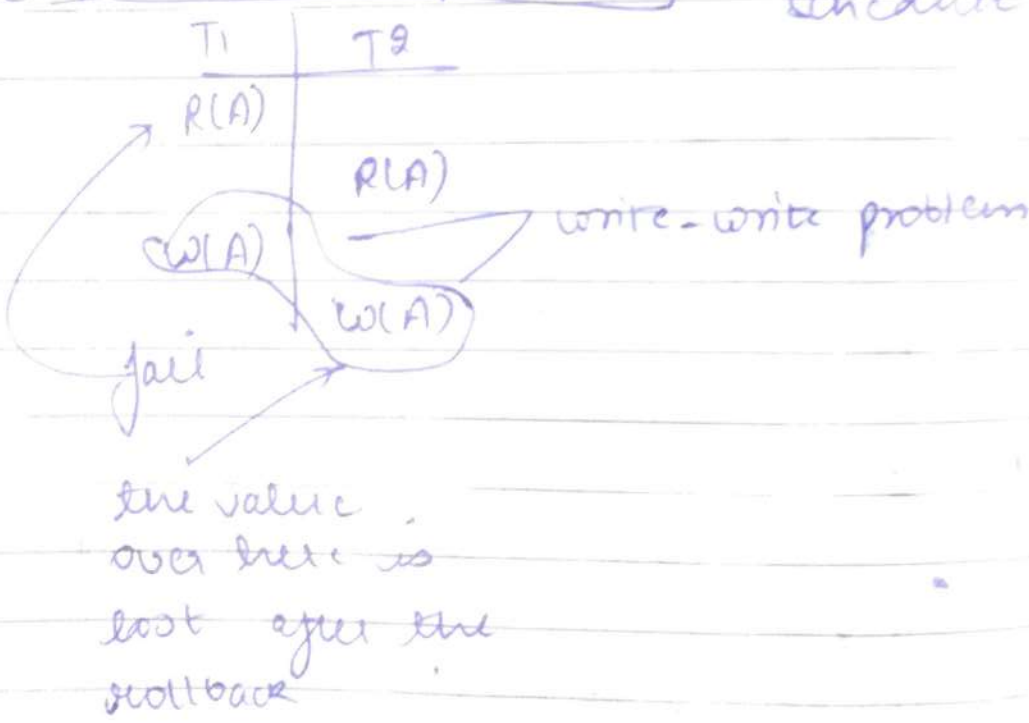
R(A)	
W(A)	
commit	
	R(A)

commit before reading A again (Cascadeless schedule)

① ~~no read after write A~~

① no read after write A till the transaction is committed.

⑤ Write-write problem → disadv of Cascadeless schedule



Concurrency Control Protocols → to control the concurrency of transactions

(a) Lock Based
(b) Time stamp based
(c) validation

Unit-5

Lock Based Protocols-

In this any transaction cannot read or write until it acquire an appropriate lock on it.

(a) Shared lock -
- know as read only lock
- can only read by transaction

(b) Exclusive lock
- the data can be read as well as written by the transaction

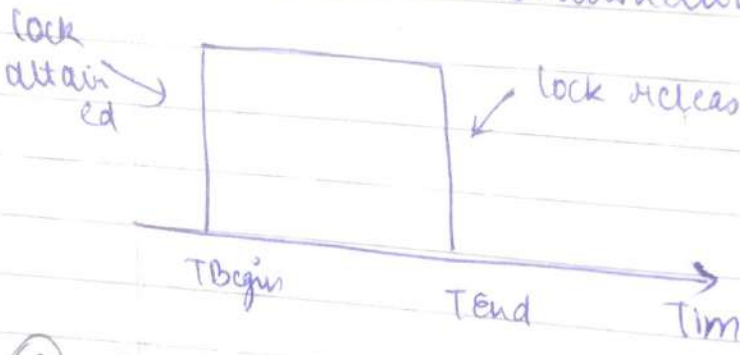
Types of locking protocols :-

(a) Simplistic lock Protocol -
- simplest way of locking data while transaction
- it allows all transactions to get the lock on the data before insert, update or delete
- it will unlock the data after completing the transaction

Rigorous 2 phase locking
MVC

(2) Pre-claiming lock Protocol :-

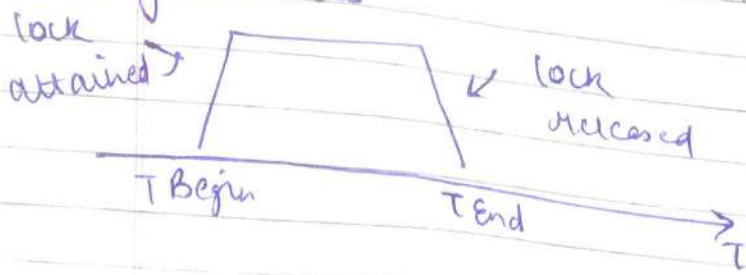
(1) It evaluates the transaction to list all the data items on which they need locks.
(2) Before initiating a transaction it requests DBMS for lock on all the items
(3) if all the locks are guaranteed it allows the transaction to occur
(4) if all the lock are ~~not~~ not guranteed then it allows the transaction to rollback



(3) Two Phase Locking Protocol -

It divides the execution phase of transaction into 3 parts

(1) When the execution of transaction starts
(2) the transaction acquires all the locks
(3) the Transaction cannot demand any new locks



2022/6/6

There there are 2 phases of 2PL
Growing Phase → a new lock may be acquired by the system but none can be released
Shrinking Phase - Existing locks held by the transaction may be released but no new lock is acquired

T1	T2
lock-S(A)	
	lock-S(A)
lock-X(B)	
—	—
Unlock(A)	
	lock-X(C)
Unlock(B)	
	Unlock(A)
	Unlock(C)
—	—

Transaction - 1

Growing phase → 1 to 3
Shrinking Phase - 5 to 7
Lock Point - 3

Transaction T2

Growing Phase → 2 to 6
Shrinking Phase → 8 to 9
Lock point - 6

2022/8/6 20.7

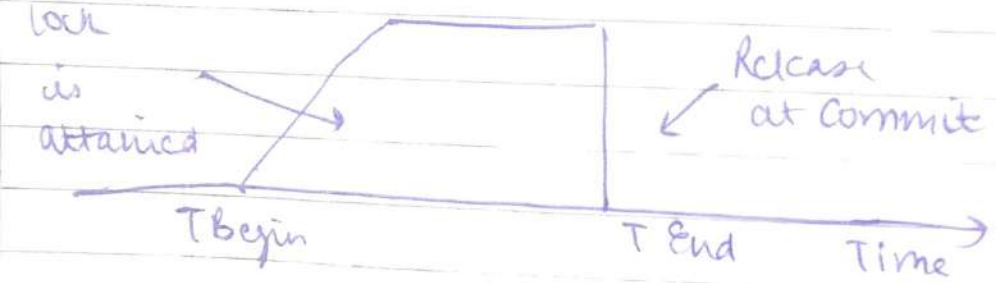
④ Strict two Phase Locking (2PL)

① In first phase, after acquiring all the locks the transaction continues to execute normally

② The only difference between 2PL and strict 2PL is that it doesnot release the lock after using it.

③ it waits for the whole transaction to commit to release all the lock

④ doesnot have shrinking phase



⑤ Conservative 2PL

① also known as static 2PL

② it requires to lock all the items before the transaction begins execution by predeclaring read set & write set.

⑥ Rigorous 2PL

same as strict

→ Validation Based Protocol :-

① Validation phase is also known as optimistic concurrency control technique

It has 3 phases :-

ⓐ Read Phase -

The transaction T_i is read & executed

ⓑ Validation Phase -

the temporary variable value is evaluated against actual value to see if it violates serializability

ⓒ Write Phase -

If the transaction is validated, then the temporary results are written into the database or otherwise the transaction is rolled back

Each phase has 3 timestamps :-

Start (T_i) → contains the time when T_i started the execution

Validation (T_i) → Contains the time when T_i finishes read phase & starts its validation process.

Finish(T_i) → contains the time when T_i finishes the write phase

This protocol is used to determine the timestamp for the transaction for serialization using timestamp of validation phase, as its actual phase where

→ Multiple Granularity –

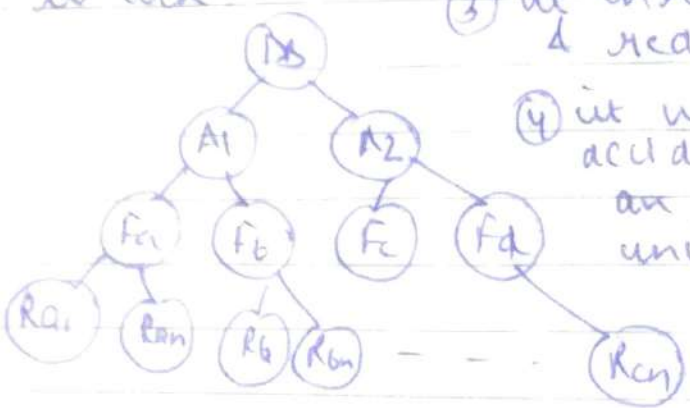
① It can be defined as hierarchy breaking the database into blocks which can be locked.

② It maintains track of what to lock & how to lock.

③ it enhances concurrency & reduces overhead

④ it makes easy to add de what to lock an item & when to unlock an item

Eg –



Highest level shows the database second level and the level below are files, records & fields

Intension Mode Lock –

(a) Intension Shared – (IS) →

it contains explicit locking at the lower level of the tree but only with shared locks

(b) Intension – Exclusive (~~IE~~ IX)

it contains explicit locking at the lower level of the tree but only with exclusive or shared locks.

(c) Shared & Intension Exclusive (SIX)

In this lock, the node is locked exclusive

shared mode & some node is locked in exclusive mode by the same transaction.

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	X
IX	✓	✓	X	X	X
S	✓	X	✓	X	X
SIX	✓	X	X	X	X
X	X	X	X	X	X

compatability matrix with intension locks

→ Thomas Write Rule →

① if TS(T) < R_TS(x)
then transaction T is aborted & rolled back & the operation is rejected

② if TS(T) < W_TS(x)
then don't execute the W_item(x) oper^n of the transaction & continue processing

③ if neither ① nor ② occurs then allowed to execute the WRITE operation by Transaction Ti & set W_TS(x) to TS(T).

→ Timestamp (from SIR)

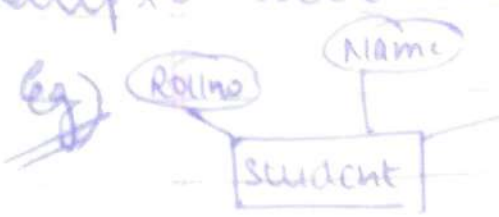
→ Problem in database (Sir notes)

2022/6/6

ER diagrams into tables

① A strong entity set with only simple attributes will require 1 table

Eg)



→

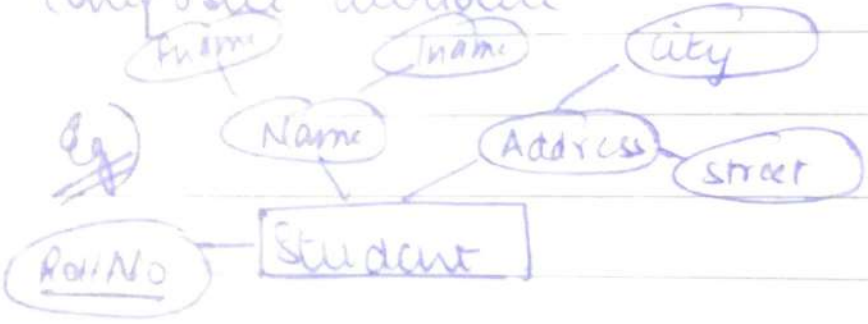
RollNo	Name	Gender

Student (RollNo, Name, Gender)

② Strong entity having composite attributes only one table

In table we take only simple attribute of composite attribute

Eg)



RollNo	Fname	Lname	City	Street

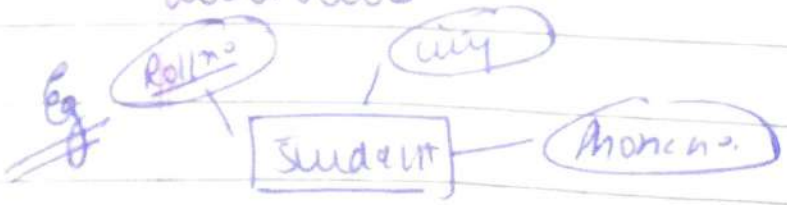
Student (RollNo, Fname, Lname, City, Street)

③ Strong entity with multivalued attribute make another table.

① One table will contain all the simple attributes with the primary key

② Other table will contain the primary key & all multivalued attributes

Eg)

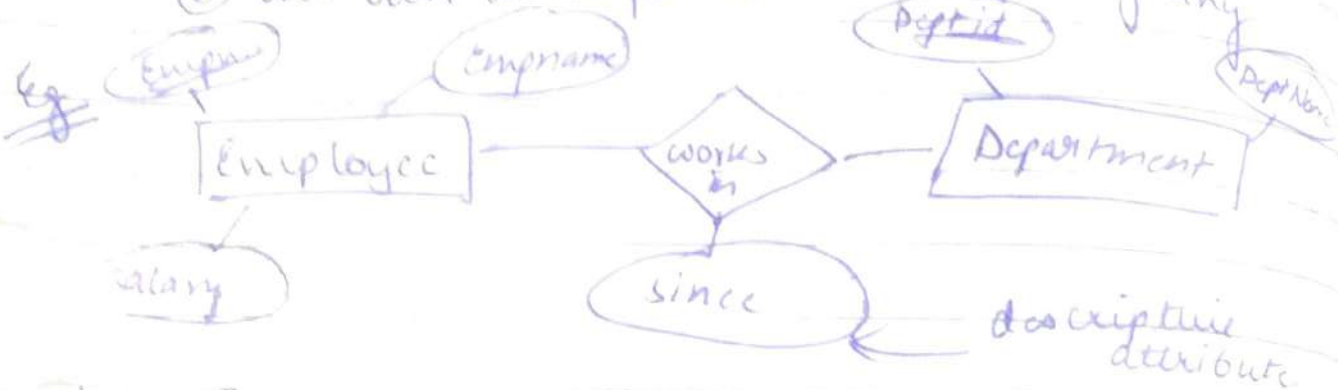


RollNo	City

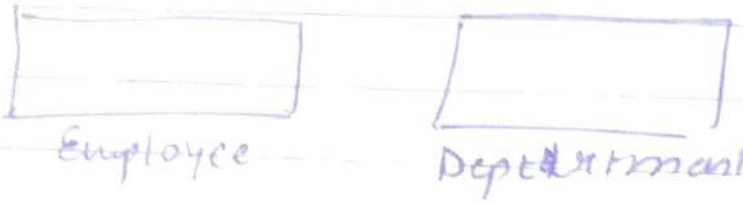
RollNo	PhoneNo.

2022:06:6 20:5

④ A relationship set will require one table
© Non descriptive attribute will be pr...
The table will contain →
ⓐ Primary key attributes of the participating entity sets
ⓑ its own descriptive attributes if any

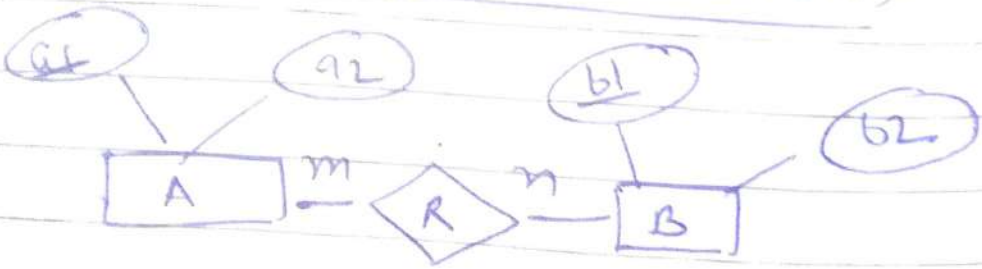


total 3 tables
① One table for Employee
② one table for work in
③ one table for Department



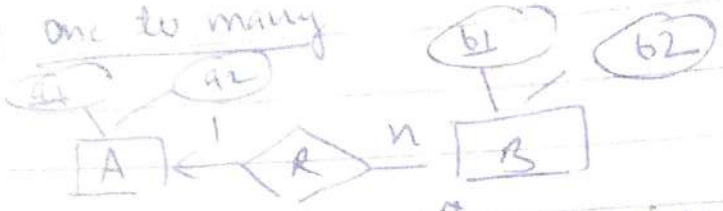
⑤ For Binary Relationships with Cardinality Ratios

① Cardinality ratio → (m:n)



A(a1, a2)
B(b1, b2)
~~R(a1, b1)~~ R(a1, b1)

(ii) one to many



multiple tables of B participate

- B and R will merge (many side entity will merge with relationship)

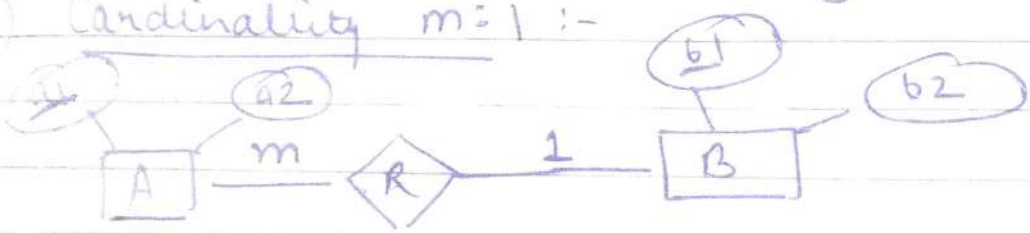
A(a1, a2)

BR(b1, b2, a1) { R ∪ B } → { b1, b2 }

{ a1, b1 }

fk ↑ not ↑ primary key

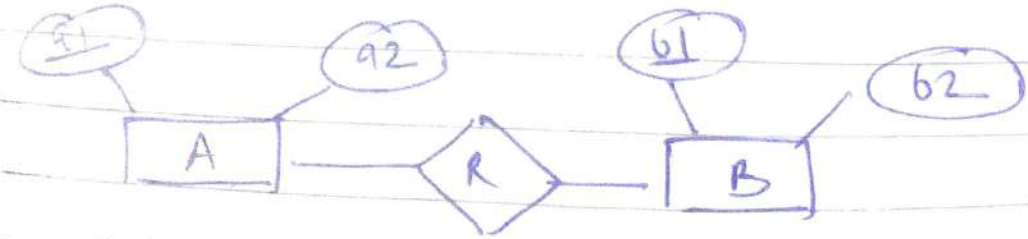
(iii) Cardinality m:1 :-



1) AR(a1, a2, b1) (same as in one to many)

2) B(b1, b2)

(iv) 1:1 :-



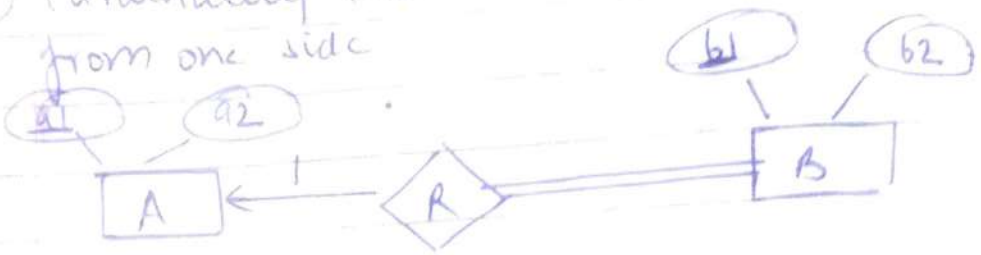
minimum 2 tables combine R with either A or B

Way 1 → AR(a1, a2, b1)
B(b1, b2)

Way 2 - A(a1, a2)
BR (b1, b2, a1)

⑥ For Binary Relationship with both cardinality & participation constraints

① Cardinality constraint + Total Participation from one side



A(a1, a2)

BR(b1, b2, a1)

Constraint → foreign key value cannot be NULL

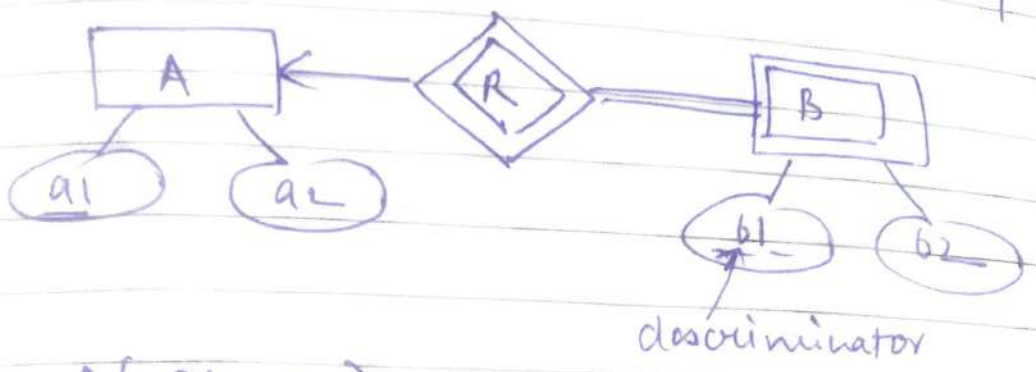
② Cardinality constraint + total participation from both sides

Only one table

ARB(a1, a2, b1, b2)

depends on
strong entity

(7) binary relationship with weak entity

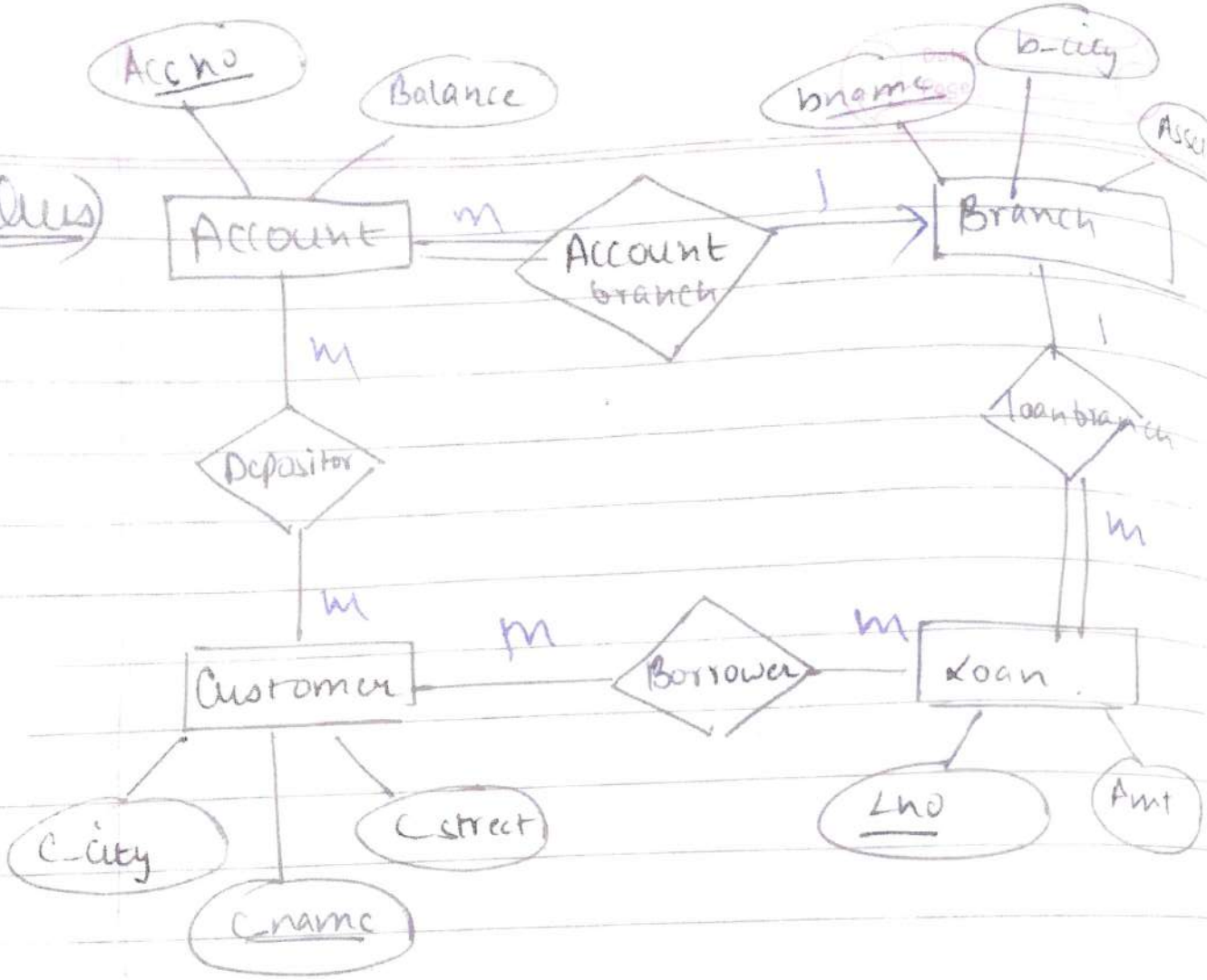


A(a1 , a2)

BR(a1 , a1 b1 , b2)

descriminator will become primary key

Ques)



Account(Accno, Balance, bname)

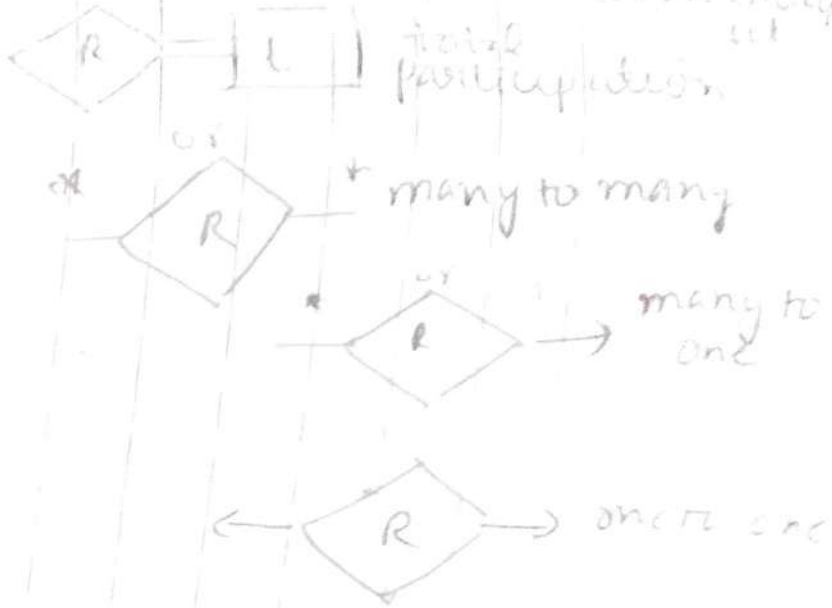
Branch(bname, bcity, Assets)

Loan (Lno, Amt, bname)

Borrower(Cname, Lno)

Customer(Cname, Cstreet, Ccity)

Depositor (Cname, Accno)


R
total participation
or
R
many to many
R
many to one
R
one to one

Transaction -

A transaction is a set of logically related operations

It contains a group of tasks

A single user performs operations for accessing the contents of the database

Operation

Read(x)

used to read value from the database and store it in a buffer in the main memory

Write(x)

used to write a value back to database from memory

→ Transaction Property :

Atomicity
Consistency
Isolation
Durability

(a) Atomicity →

all transactions take place at once if not aborted

Two operations

a) Abort → if a transaction aborts then the changes made are not visible

b) Commit → if a transaction commits then the changes made are visible

(b) Consistency →
The integrity constraints are maintained so that the database is consistent before and after transaction.

Eg → Total before T occurs → 600 + 300 = 900
Total after T occurs → 500 + 400 = 900

(c) Isolation -
It shows that the data inconsistency used at the time of execution of a transaction cannot be used by second transaction until the 1st is completed

(d) Durability -
If a transaction updates the data and commits then the database will hold the modified data
If a transaction commits but the system fails before the data could be written into the disk then the data will be updated once the system springs back to action

That means the database should be durable enough to hold all its latest updates even if system fails or restarts

→ States of Transaction:

ⓐ Active State –
In this state the transaction is being executed.
Insertion, deletion or updating a record in the table is done here. But all records are still not saved to the database.

ⓑ Partially Committed –
A transaction executes its final operation but the data still not saved to the database

ⓒ Committed –
All the operations are executed successfully and saved permanently

ⓓ Failed State –
If any of the checks made by the database system fails then the transaction is said to be in failed state.

(e) Aborted -

If any of the checks fail and the system has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not it will abort or rollback the transactions to bring the database in consistent state.

After aborting the transaction the database recovery module will select one of the 2 operations
1) Restart the transaction
2) Kill the transaction.

SELECT Query : The SELECT Statement in SQL is used to retrieve or fetch data from a database.

SELECT column1,column2 FROM table_name column1 , column2: names of the fields of the table table_name

To fetch the entire table or all the fields in the table:

SELECT * FROM table_name;

Distinct Clause : When we use distinct keyword only the unique values are fetched.

SELECT DISTINCT column1, column2 FROM table_name

INSERT INTO Clause : The INSERT INTO statement of SQL is used to insert a new row in a table.

INSERT INTO table_name (column1, column2, column3,..) VALUES (value1, value2, value3,..);

DELETE Statement : The DELETE Statement in SQL is used to delete existing records from a table.

DELETE FROM table_name WHERE some_condition;

some_condition: condition to choose particular record.

UPDATE Statement : The UPDATE statement in SQL is used to update the data of an existing table in database.

UPDATE table_name SET column1 = value1, column2 = value2,. .. WHERE condition;

ORDER BY : The ORDER BY statement in SQL is used to sort the fetched data in either ascending or descending according to one or more columns.

SELECT * FROM table_name ORDER BY column_name ASC|DESC;

| : use either ASC or DESC to sort in ascending or descending order

Wildcard characters : A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the LIKE operator.

SELECT column1,column2 FROM table_name WHERE column LIKE wildcard_operator;

Wildcard operators :

% : used in substitute of zero or more characters

_ : used in substitute of one character

Join (Inner, Left, Right and Full Joins) : SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.

INNER JOIN : The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied.

SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 INNER JOIN table2 ON table1.matching_column = table2.matching_column;

LEFT JOIN : This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join.

SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 LEFT JOIN table2 ON table1.matching_column = table2.matching_column;

RIGHT JOIN : This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join.

SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 RIGHT JOIN table2 ON table1.matching_column = table2.matching_column;

FULL JOIN : FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables.

SELECT table1.column1,table1.column2,table2.column1,.... FROM table1 FULL JOIN table2 ON table1.matching_column = table2.matching_column;

Group by : The GROUP BY statement groups rows that have the same values into summary rows.

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

SELECT column_name(s) FROM table_name WHERE condition GROUP BY column_name(s) ORDER BY column_name(s);

The SQL commands are mainly categorized into five categories as:

Data Definition Language (DDL):

DDL commands are:
CREATE,DROP,ALTER,TRUNCATE,COMMENT,RENAME

Data Query Language (DQL) : select

Data Manipulation Language (DML) : Insert , update ,delete, lock

Data Control Language (DCL) : Grant , revoke

Transaction Control Commands (TCL) : Commit , rollback , savepoint , set transaction