Page No:
Date:

Introduction to database systems

- **Data**
  - Raw facts, unprocessed facts
  - Refers to what is actually stored

- **Information**
  - Result of processing raw data
  - Refers to meaning of the data understood by the user

Data management focuses on the generation, storage and retrieval of data.

- **Limitations of file processing systems**
  - Redundancy problems
    - Repetitive data
  - Data inconsistency problems
    - Incorrectness of data
  - Lack of data integration
    - Complex and time consuming

- **Database**

A database is a collection of related data which represents some aspect of the real world. A database system is designed to be built and populated with data for a certain task.

- **Database Management System (DBMS)**

DBMS is a software for storing and retrieving user's data while considering appropriate security measures. It consists of a group of programs which manipulate the database. The DBMS accepts the request for data from an application and instructs the operating

Page No:
Date:

Types of databases:

Based on the number of users using it

i) Single User Database System
(At one time only a single user can interact with the database)

ii) Multi User Database System
(At one time multiple users can interact)

Based on the location of the database

i) Centralized Database System
(Data located at a single place)

ii) Distributed Database System
(Data distributed across different locations)

Advantages of DBMS over file system

(i) Restricting unauthorized access
(ii) Providing multi user interface
(iii) Providing backup and recovery
(iv) Allows data sharing
(v) Enforcing integrity constraints
(vi) Solving data isolation
(vii) Controlling redundancy & inconsistency
(viii) Providing economies of scaling

Data Model

Data Model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints. It provides a way to describe the design of database

Components of data model

(i) Entity: Anything that exists, living or non living but not imaginary.

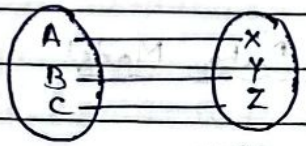
(ii) Entity Set: Set of entities of the same type

(iii) Attributes: Characteristics of an entity

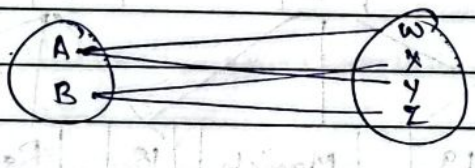
(iv) Constraints: Restrictions placed on a data

(v) Relationships: Similarities or a connection b/w entities

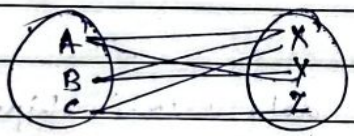
One to One (1:1) Relationship



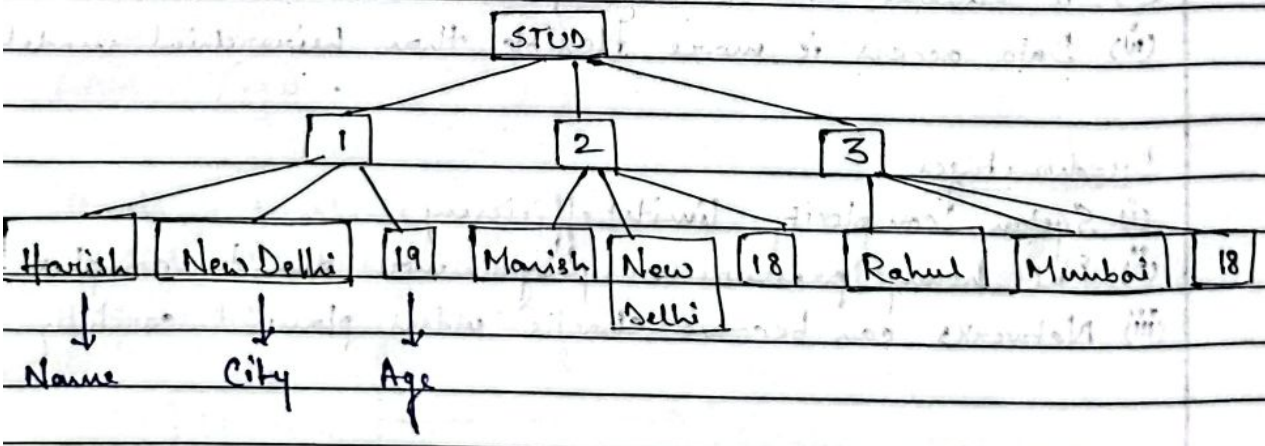
One to Many (1:M) Relationship



Many to Many relationship



Hierarchial Data Model

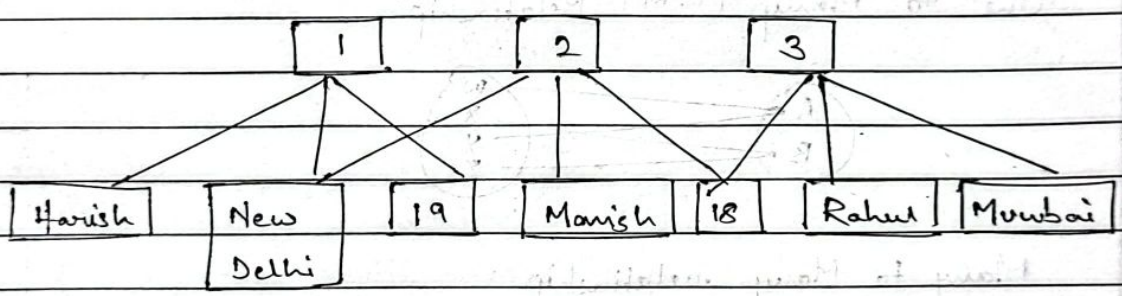


Page No:
Date:

Advantages
(i) It is efficient with 1:M relationships
(ii) Efficient storage for data that have clear hierarchy
(iii) It promotes data sharing

Disadvantages
(i) It is complex to implement and difficult to manage
(ii) It can't represent M:N (many to many) relationships
(iii) There is no DDL and DML

Network Data Model



Advantages
(i) It represents complex data relationships better than hierarchial model
(ii) It includes DDL and DML
(iii) It handles M:N relationships
(iv) Data access is more flexible than hierarchial model

Disadvantages
(i) System complexity limits efficiency
(ii) Put heavy pressure on programmers due to complex structure
(iii) Networks can become chaotic unless planned carefully

Page No:
Date:

Relational Data Model

STUD

roll	Name	City	Age
1	Harish	New Delhi	17
2	Manish	New Delhi	18
3	Rahul	Mumbai	18

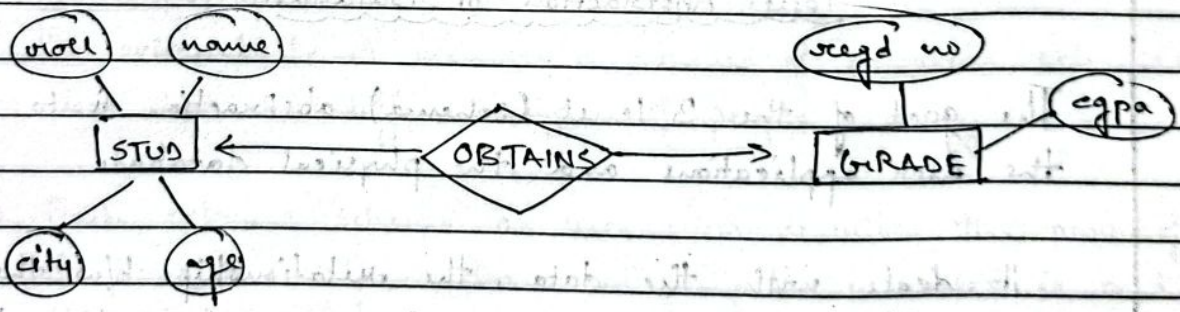
Advantages

(i) Tabular view substantially improves conceptual simplicity
(ii) Changes in table's structure do not affect data access or application programs
(iii) RDMS isolates the end users from physical level details and improves implementation and management simplicity

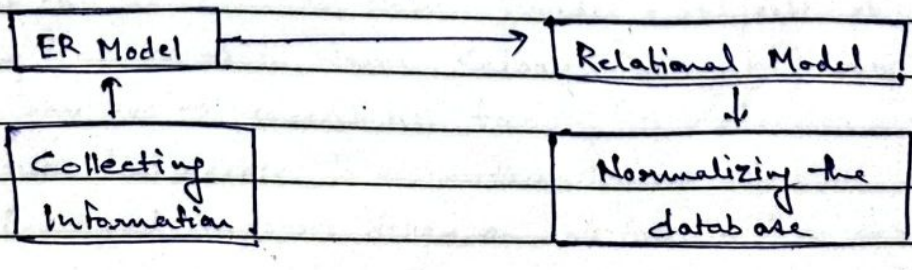
Disadvantages

(i) Conceptual simplicity gives relatively untrained people the tools to use a good system poorly.
(ii) It may promote islands of information problems as individuals and departments can easily develop their own applications.

Entity Relationship (ER) Model



It is a graphical representation of entities and their relationships in a database structure. ER models are basically represented using ER diagram.



Page No:
Date:

Advantages

i) Visual representation makes it an effective communication tool
ii) It is integrated with dominant relational model
iii) Visual modelling yeilds exceptional conceptual simplicity

Disadvantages

i) There is limited constraint representation
ii) There is no DML
iii) There is limited relationship representation.
iv) Loss of information content when attributes are removed from entities to avoid crowded displays.

Explain database architecture (diagram)

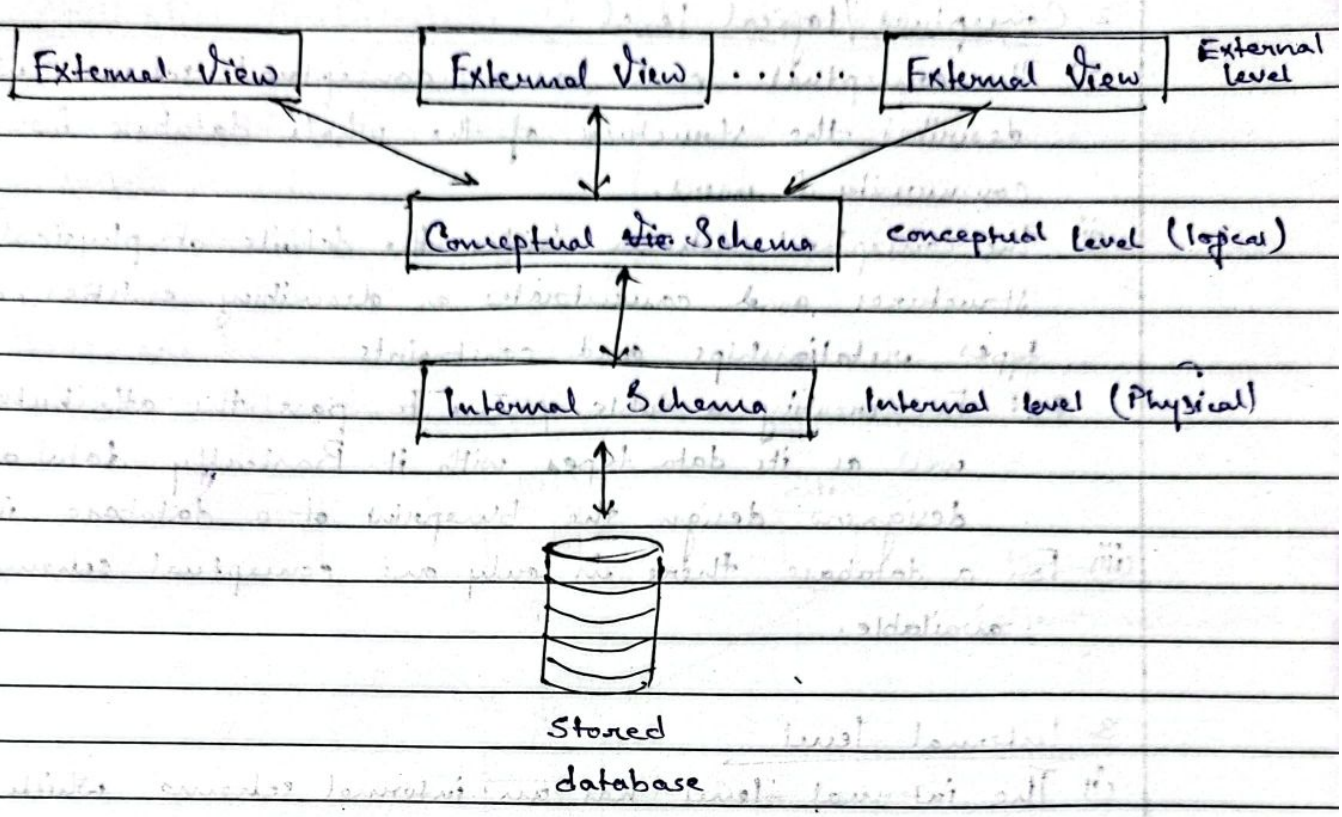
Components of DA
1. Users & DBA
2. Query processor
3. Storage manager
4. Disk storage

Level Abstraction of Database

The goal of the 3 level (schema) abstraction is to separate the user applications and the physical database.

It deals with the data, the relationship b/w them and the different access methods implemented on the database

The logical ~~diagram~~ design of a database is called as schema.



User knows that their data is stored somewhere but they don't know how or where the data is stored. It means the process is kept hidden or abstracted from the user. To achieve this the three level abstraction concept was introduced.

1. External/View level

(i) The external level includes a number of external schemas or user views for different types of people accessing it.

(ii) Each external schema or user view describes the part of the database that a particular user group is interested in and hides the details ~~of the~~ from other users.

Ex: Suppose in a database of university there are three external schemas → Student, Faculty, Dean

Student can see → Roll no, Name, Grades, Subjects etc

Faculty can see → id, Name, Salary, Date of joining etc

Dean can see → student list, faculty list, attendance etc.

* Each user has access to ~~each other~~ different types of data but not each others. Front end developers design an interface in this stage for the user

2. Conceptual / logical level

(i) The conceptual level has a conceptual schema which describes the structure of the whole database for a community of users.

(ii) The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships and constraints

Ex: For creating a table you've to pass the attributes as well as its data types with it. Basically database designers design the blueprint of a database in this stage

(iii) For a database there is only one conceptual schema available.

3. Internal level

(i) The internal level has an internal schema which describes the physical storage structure of the database system.

(ii) Like conceptual schema, there is only one internal/physical schema available

(iii) It is the one which is ~~available~~ closest to the physical storage. In this stage Database Administrators have the access to the whole database and he/she decides where the files should be stored (centralized or decentralized database)

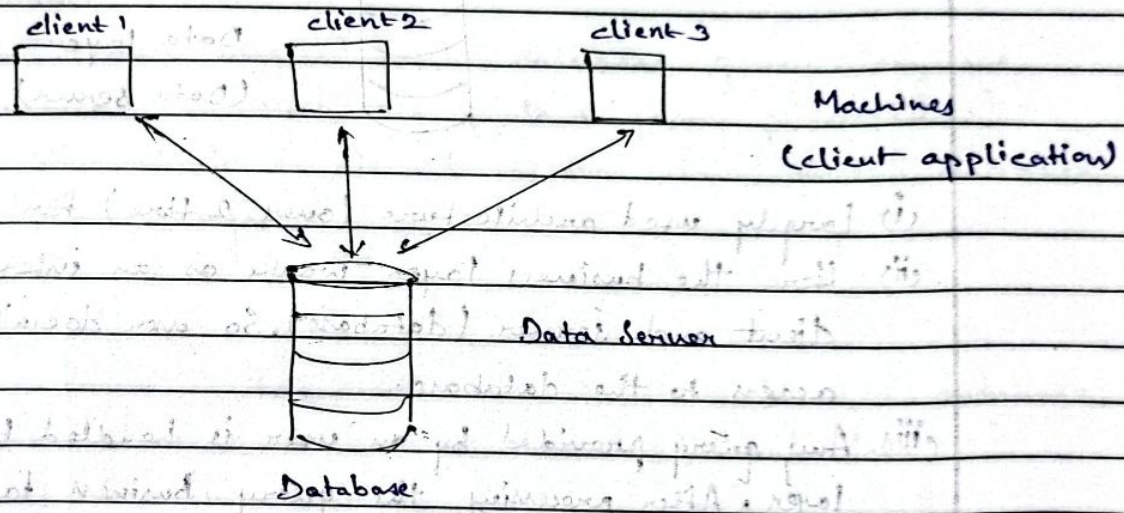
When an user is seeing a data fetched from the database they generally see it in the tabular form. But data aren't actually stored in the database as tabular form. Actually the data's are stored as files and when an user needed it, the file is just converted into tabular format.

Page No:
Date:

Application Architecture Used for Database

(i) Client machines are ~~there~~ those on which the remote database system runs.
(ii) Server machines are those on which the database system runs.

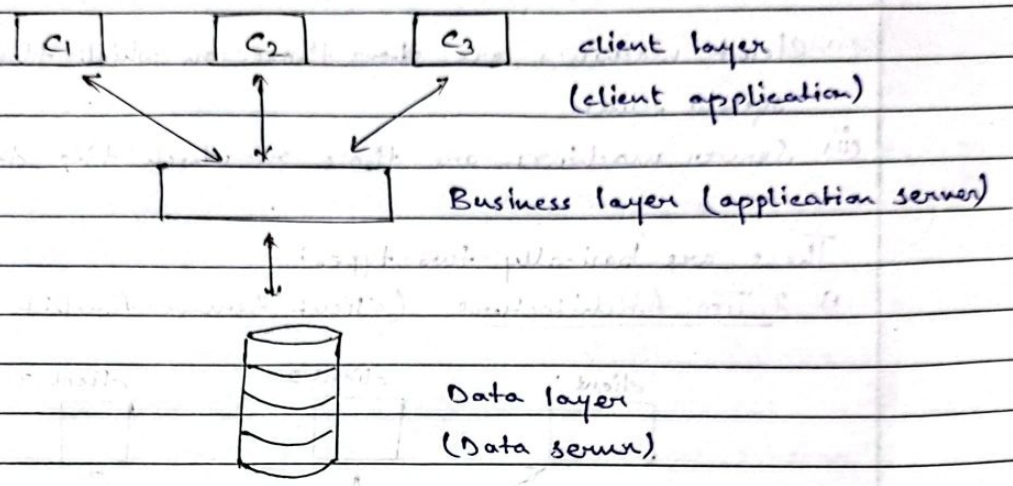
There are basically two types:
1) 2 Tier Architecture (Client Server Architecture)



i) Here the client machine runs an interface or applicatin that helps an user to establish a connection b/w the database and the machine.
ii) User simply writes a query (application program) through the interface and the database then processes it and returns the required data to the client.
iii) Maintainance is easy because of limited and authorized clients.
iv) Fails in terms of scalability. (handling large no. of clients, providing data 24x7)
v) Here user has direct access to the database which can create a huge problem. Fails in terms of security.

Page No:
Date:

2) 3 Tier Architecture



(i) Largely used architecture (over 2 tier) for its advantage.
(ii) Here the business layer works as an intermediate b/w the client and server (database). So user doesn't have direct access to the database.
(iii) Any query provided by an user is handled by the business layer. After processing the query business layer wants the required data from the server and returns it to the user. (In 2 tier all the processes and data transfer things are handled by the server. So its also an disadvantage for 2 tier architecture that it may malfunction because of too much load)

Types of Database Users

① Naive Users

- They are the normal or sophisticated users who interact with the system by invoking application programs that have been written previously.

- The typical user interface for naive users is a form interface where the user can fill in appropriate fields of the form.

② Application programmers
- They are computer professionals who write application programs to access data from the database.
- Application programmers can use different tools to develop user interface.

③ Sophisticated Users
- They interact with the system without creating any application program.
- They form their request in a database query language and submit each such query to a query processor.
- Analysts who submit queries to explore data in the database fall in this category.

④ Specialized Users
- They are sophisticated users who write specialized database applications that don't fit into the traditional data processing framework.

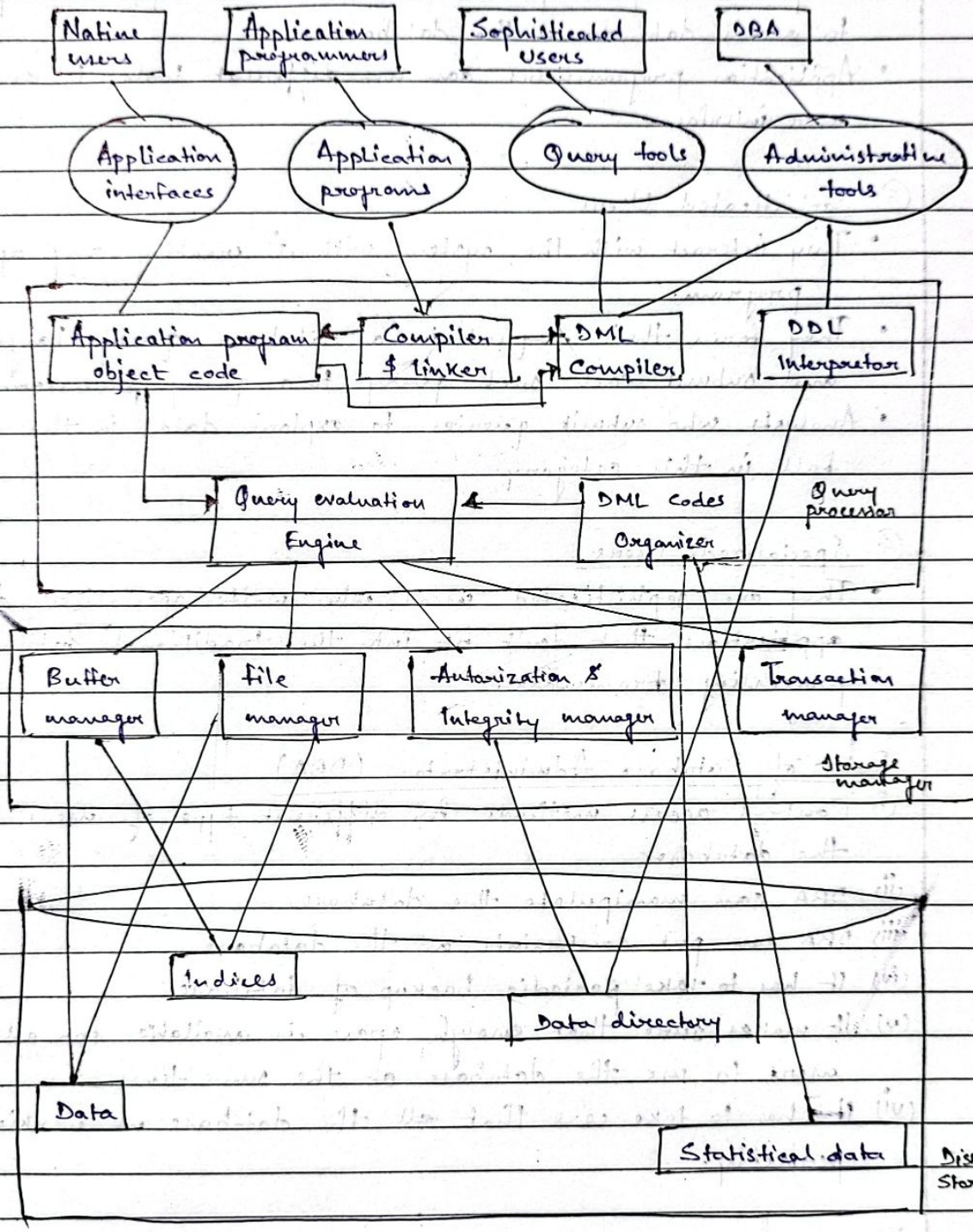
Roles of Database Administrator (DBA)
(i) Control access methods for different type of users to use the database.
(ii) DBA can manipulate the database
(iii) DBA can put constraints on the database
(iv) It has to take periodic backup of database
(v) It makes sure that enough space is available for all the users to use the database at the same time.
(vi) It has to take care that the database is working perfectly.

Physical data independence indicates that the internal schema can be changed without any change to the conceptual schema.

Logical data independence indicates that the conceptual schema can be changed without affecting existing external schemas.

Page No:
Date:

Database Architecture



Page No:
Date:

Disadvantages of DBMS

(i) Large file size
(ii) Increased complexity
(iii) Greater impact of failure
(iv) More difficult to recover.

Chapter 2.

Page No:
Date:

Entity Relationship Model

(i) Collect information about what they want in the database
(ii) Draw the entity relation model for the requirement

Entity

An entity is a thing or object that exist in the real world

Attributes → Oval Shape ⬭
Entities → Rectangle ▭

Relationship

Association b/w entities are referred to relationship

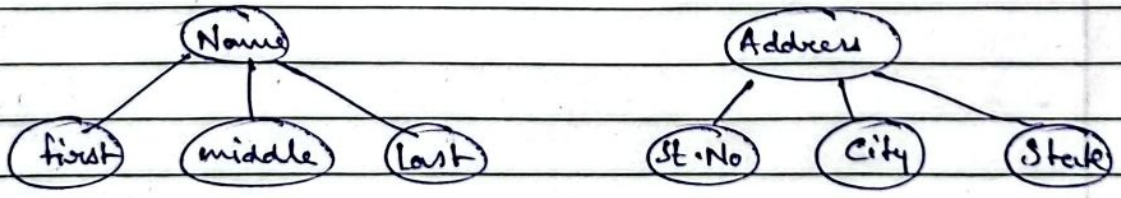
Relationship → Diamond Shape ◇

NULL Value :

When an entity doesn't have a value for it, we can put NULL value for it.

Simple & Composite Attributes

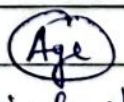
Attributes that can't be sub divided are simple attributes.
Ex: Roll no
Attributes that can be sub divided are composite attributes
Ex: Name (first and last name)



Multivalued attributes

Attributes with more than one value
Ex: Student with more than one mail and phone number

Multi valued



Single Valued

Stored & Derived Attributes

Attribute with independent existence is stored attributes.
Ex: Age of a student

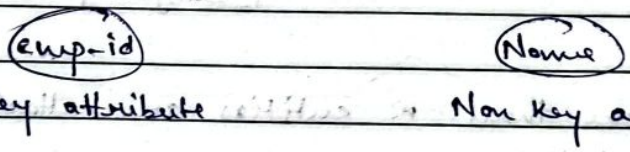
Attribute whose value is depending on other stored attribute is called as derived attribute.
Ex: DOB of a student can be derived from age

Descriptive Attributes

A relationship may also have attributes called as descriptive attribute.

Keys and Non Keys Attribute

Uniquely identified attribute → Key attribute



Complex Attribute

Composite + Multivalued Ex: Address → (current + permanent) → multi valued
→ (street name, city, state) → composite

Types of relationship

(1) M : N relationship

(2) 1 : M relationship

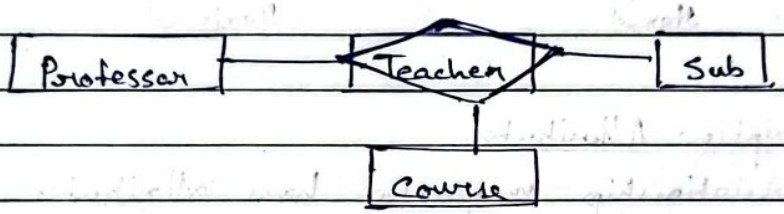
(3) 1:1 relationship

Degree of relationship type

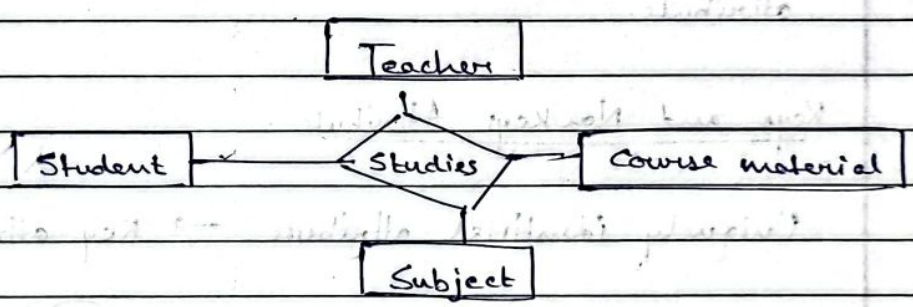
A relationship type is a meaningful association among entities

1) Binary : 2 entities are attached to an attribute

2) Ternary :

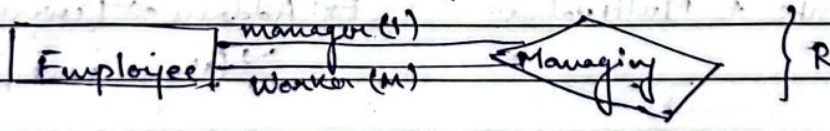


3) Quaternary :



4) n-ary : More than n entities are attached to a single attribute

5) Recursive : Special case of relationship

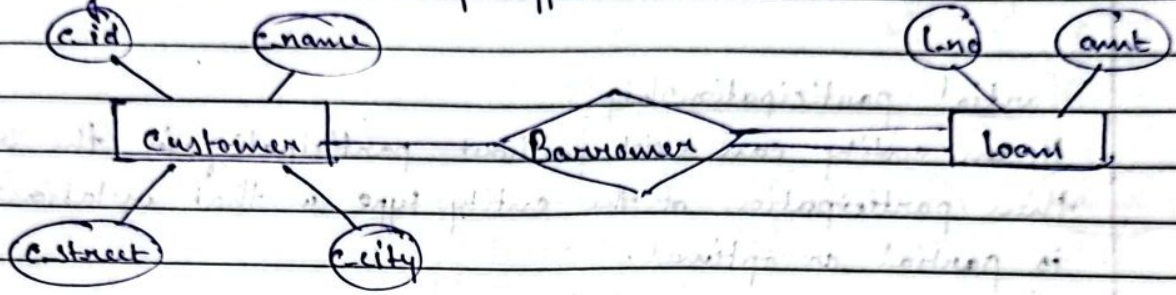


One entity has multiple roles.

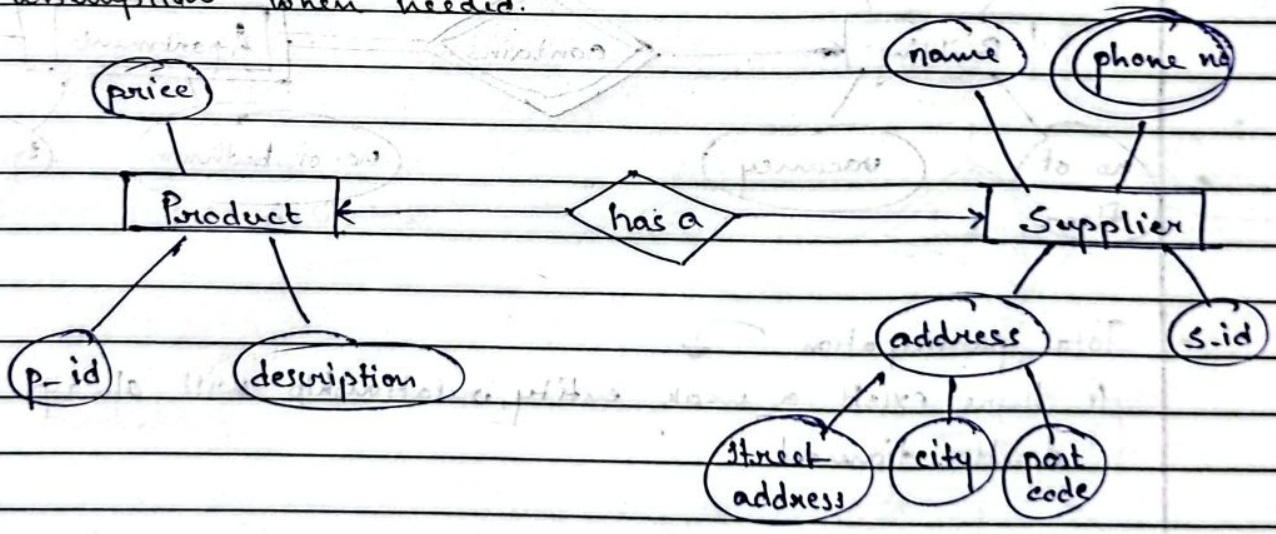
Page No:
Date:

Participation Constraints

The participation constraint for an entity set in a binary relationship type is based on whether an entity of that entity set needs to be related to an entity of the other entity set through this relationship type



Q Construct the ERD to represent information of product and supplier. Each product has a description, a price and a supplier. Suppliers have addresses, phone numbers and name. Each address is made up of a street address, a city and a postcode. Make assumptions when needed.



* s_id is taken as an attribute of the supplier and it is also the primary key

Total participationship

If in order to exist, every entity must participate in the relationship, then participation of the entity set in that relationship type is total or mandatory. It is represented by double lines

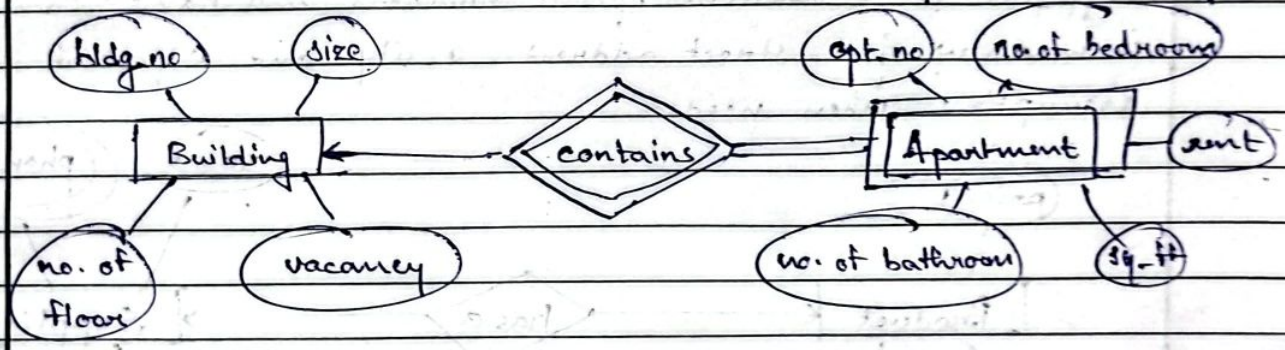
Partial participationship

If an entity can exist without participating in the relationship then participation of the entity type in that relationship type is partial or optional.

Strong & Weak Entity Set

Independent existence → strong entity

Dependent existence → Weak entity → (represented by double rectangle)



Total participation →

If there exists a weak entity, relationship will always be double diamond.

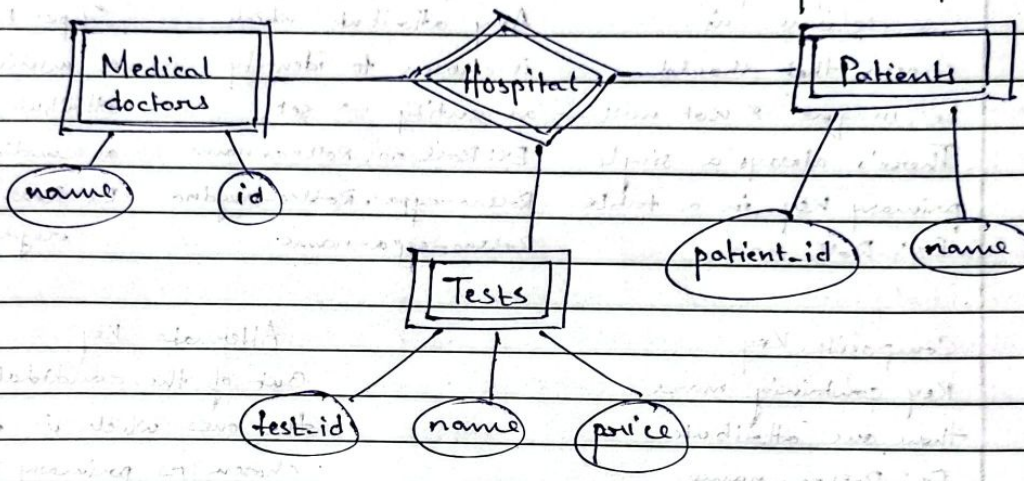
Primary Key → Primary Key of strong entity

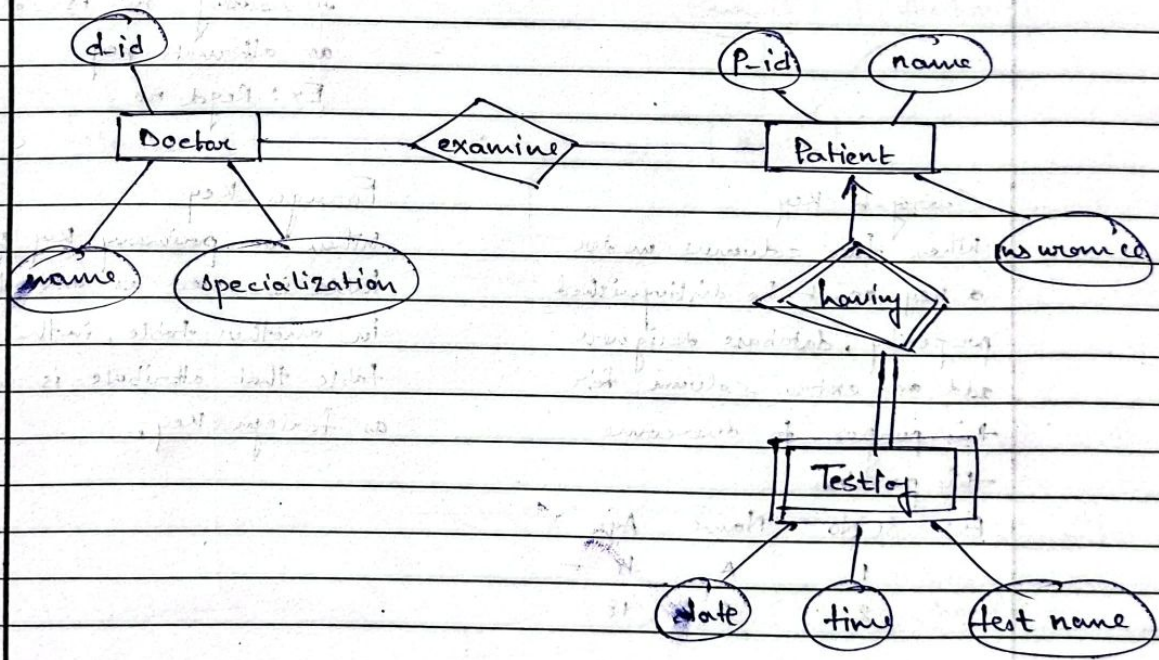
+

distributor Partial Key

Page No:
Date:

Q Construct an ER diagram with a set of patients and a set of medical doctors. Associated with each patient a log of the various tests & examinations conducted. Make neccessary assumptions.





Page No:
Date:

Keys

Primary Key
One column is chosen that should be unique & not null
There's always a single primary key in a table
Ex: Roll no

Super Key
Any attribute which is chosen to identify an entity → set
Ex: Roll no, Rollno-name
Rollno-cgpa, Rollno-regdno
Rollno-cgpa-name

Candidate Key
Super key with a minimal attribute is called as candidate key
Ex: Roll no, regd no.

Composite Key
Key containing more than one attribute
Ex: Rollno-name

Alternate Key
Out of the candidate keys the ones which is not chosen as primary key - the remaining one is called as alternate key
Ex: Regd no

Surrogate Key
When two columns under a key can't be distinguished properly, database designers add an extra column for this purpose to overcome this problem

Ex:

Sl No	Name	Age
1	A	19
2	A	18

Foreign Key
When a primary key of a table is used as an attribute in another table, in the other table that attribute is called as foreign key.

Keys for relationship

- M:N Relationship

The primary key of the relationship set contains the union of the primary keys of the entity set.

Page No:
Date:

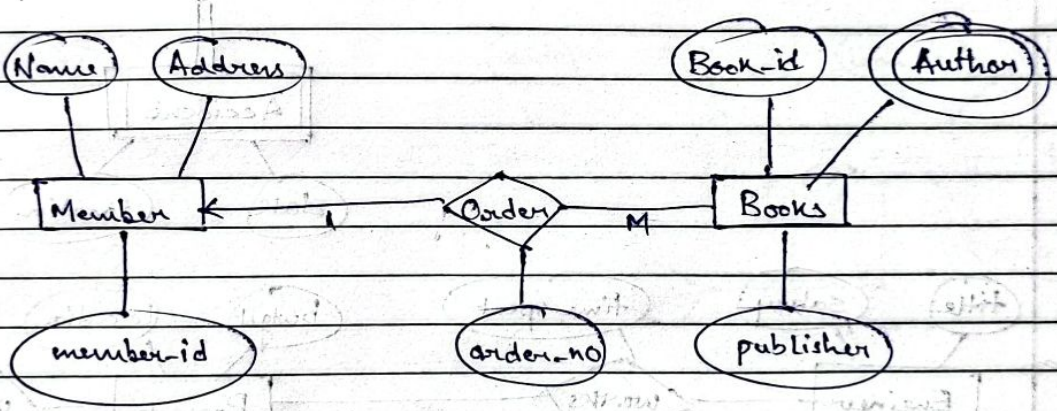
1 : M Relationship

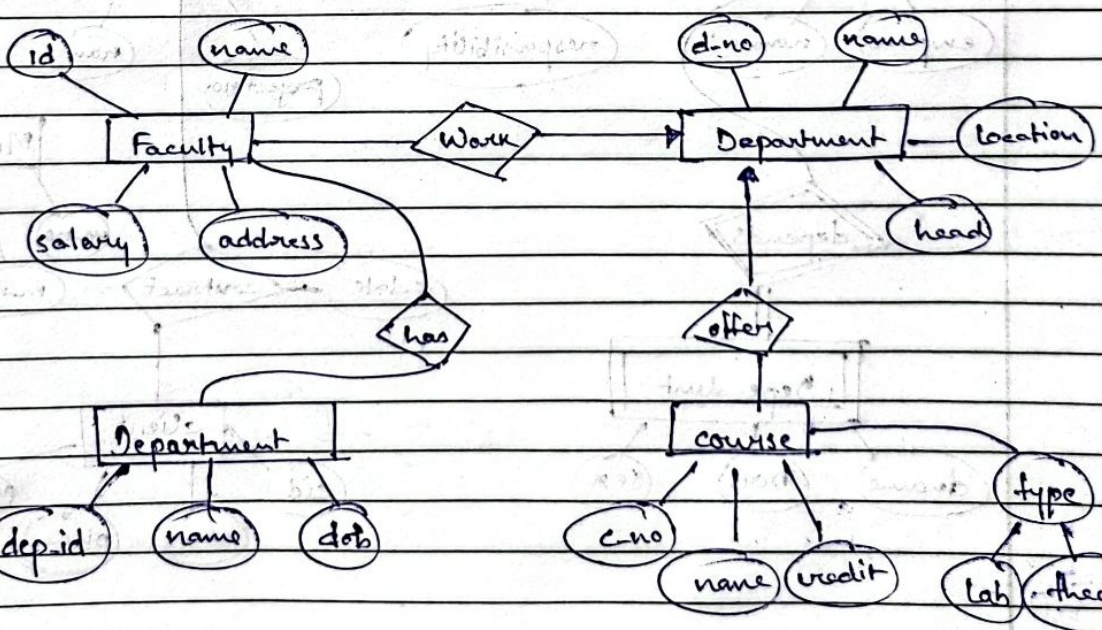
The primary key of the relationship set is the primary key of the many side entity set.

1 : 1 Relationship

The primary key of the relationship set is either the primary key of the entity set

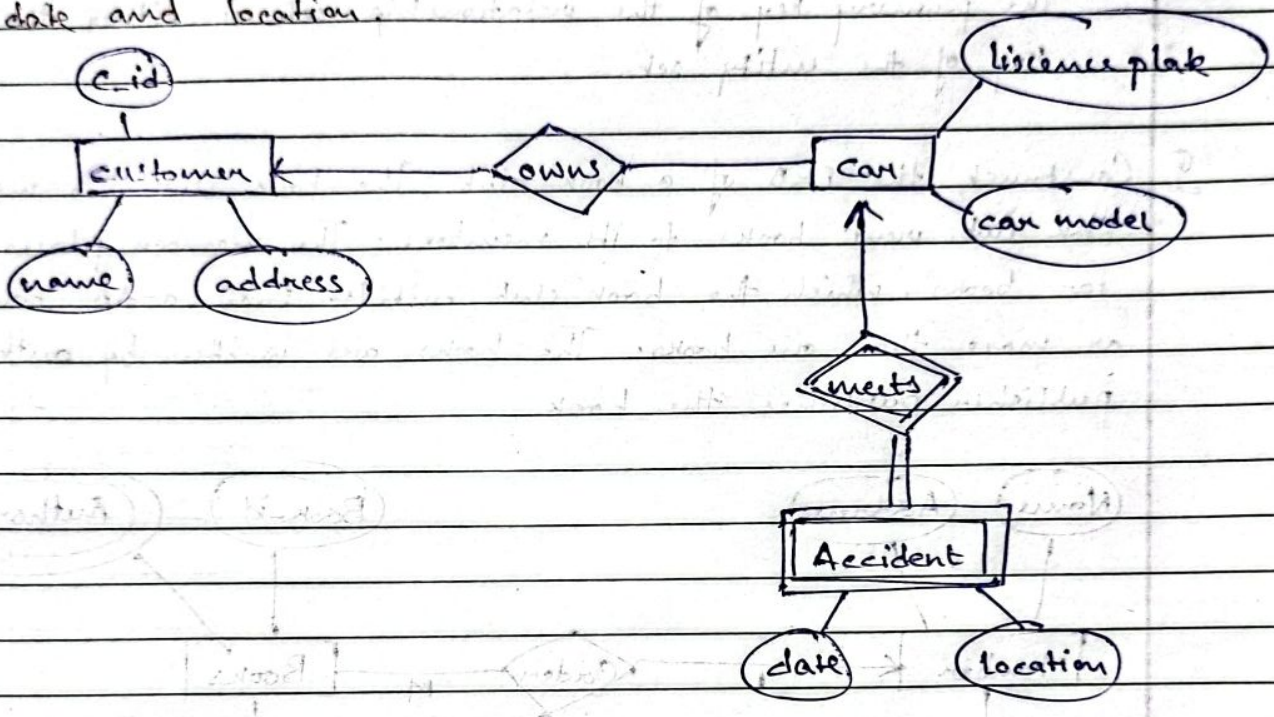
Q Construct the ERD of a book club. The book club has member. The book club rent books to its members. The member places order for books, which the book club fulfills. Each order contains one or more than one books. The books are written by authors. The publisher publishes the book.



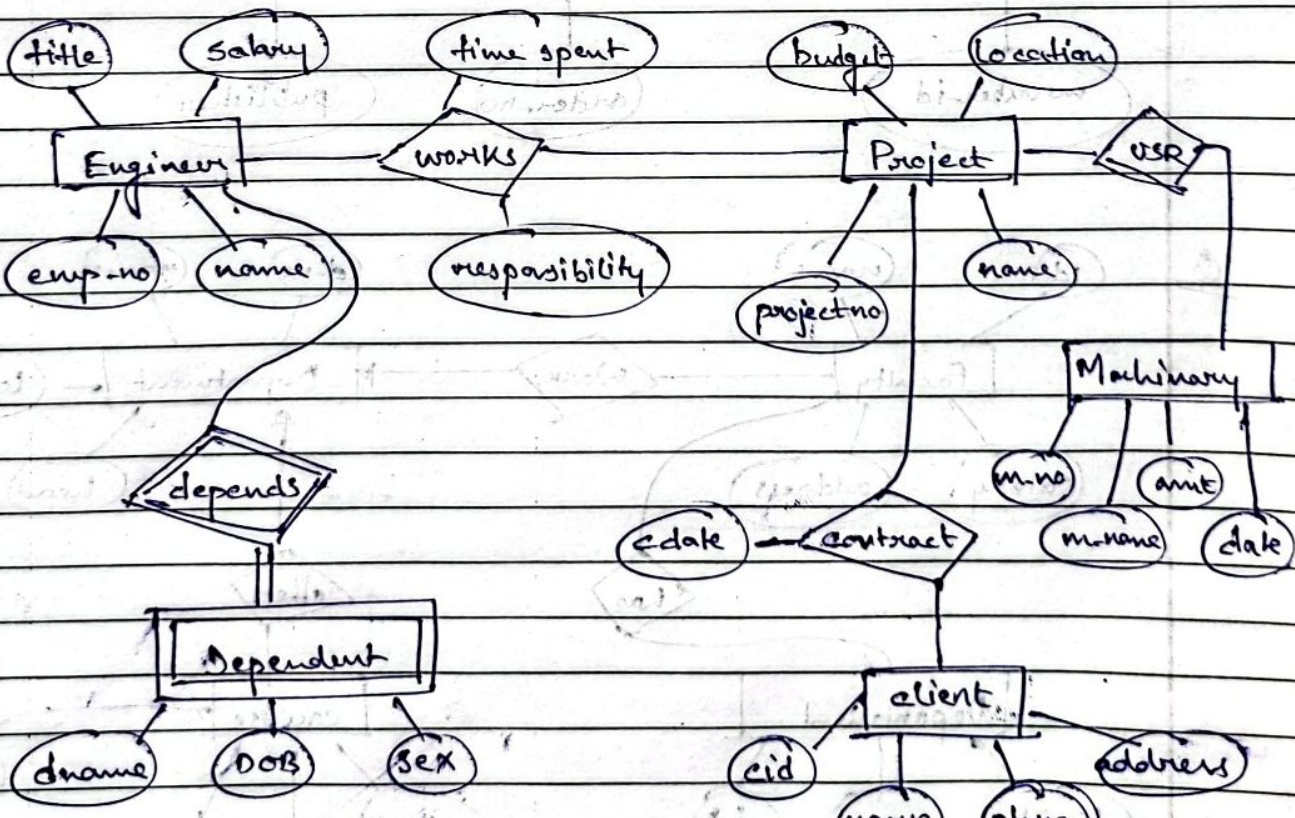


Page No:
Date:

Q Construct an ER diagram for a car insurance company where customers own one or more cars each. Customers are identified by id, name and address. Each car can be identified by car model, liscence plate, car make. Each car is associated with zero to any number of accidents. Accident has accident date and location.



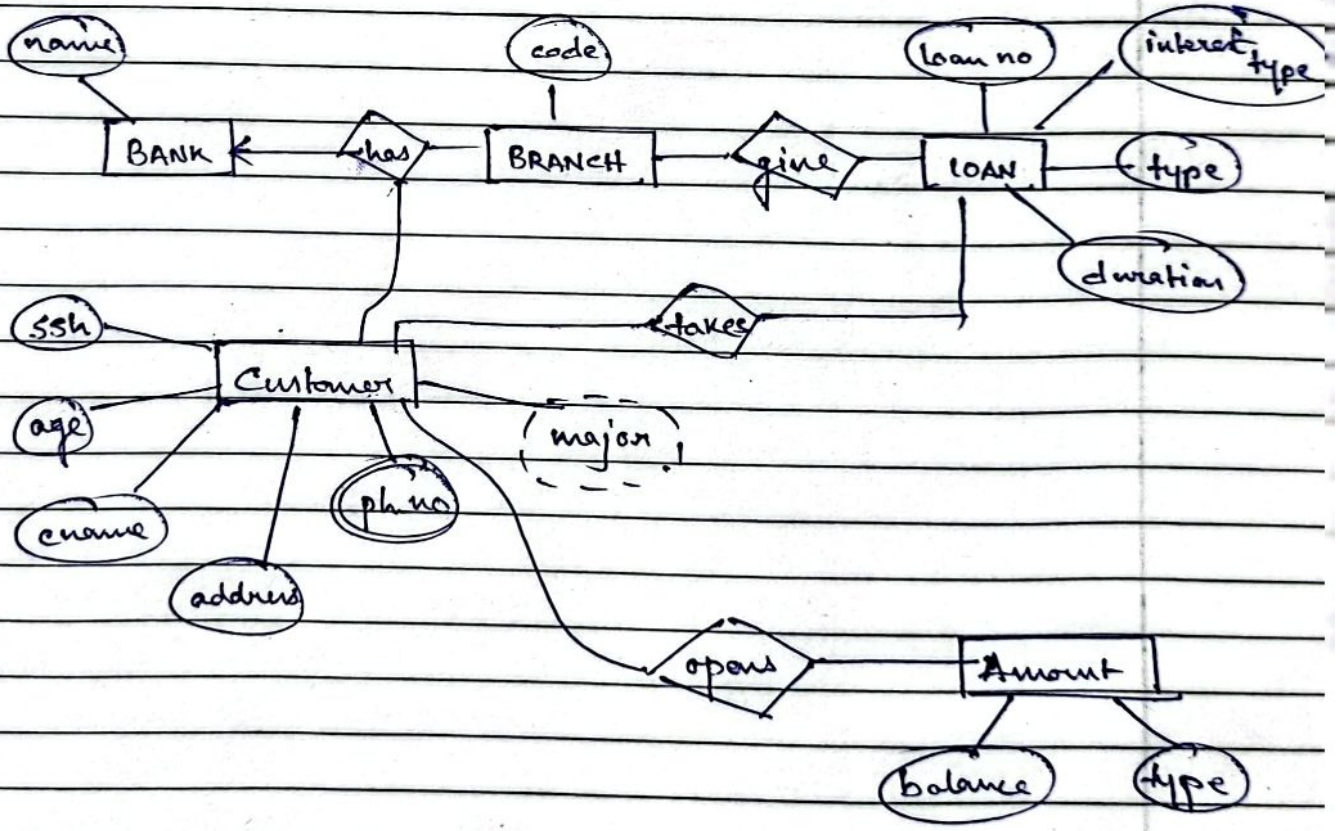
Q



Issues with ER Model

(i) Use of entity set vs attributes
(ii) Use of entity set vs relationship set
(iii) Use of binary vs nary relationship set
(iv) Placement of relationship attributes

- For one-one you can add the descriptive attribute to any side of the entity.
- For one-many you have to add the descriptive attribute to the many side entity.
- For many many you have to keep the descriptive attribute with the relationship.

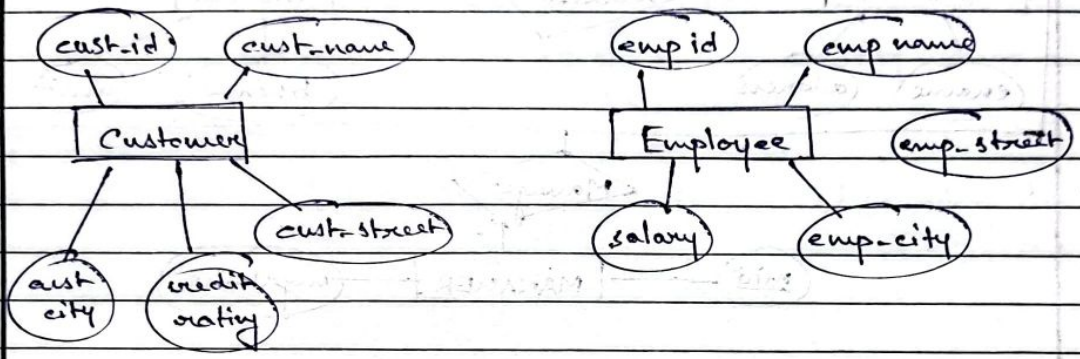


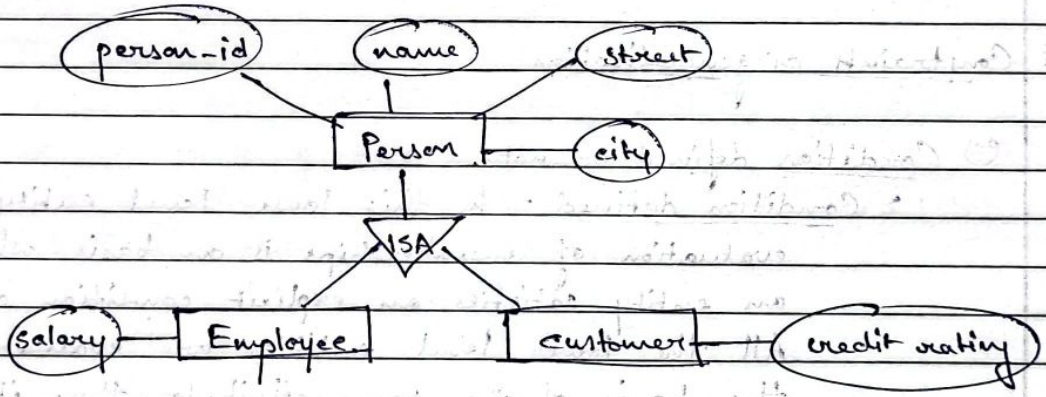
Extended ER Model

1. Specialization → ▽ ISA
2. Generalization ↗
3. Aggregation

Page No:
Date:

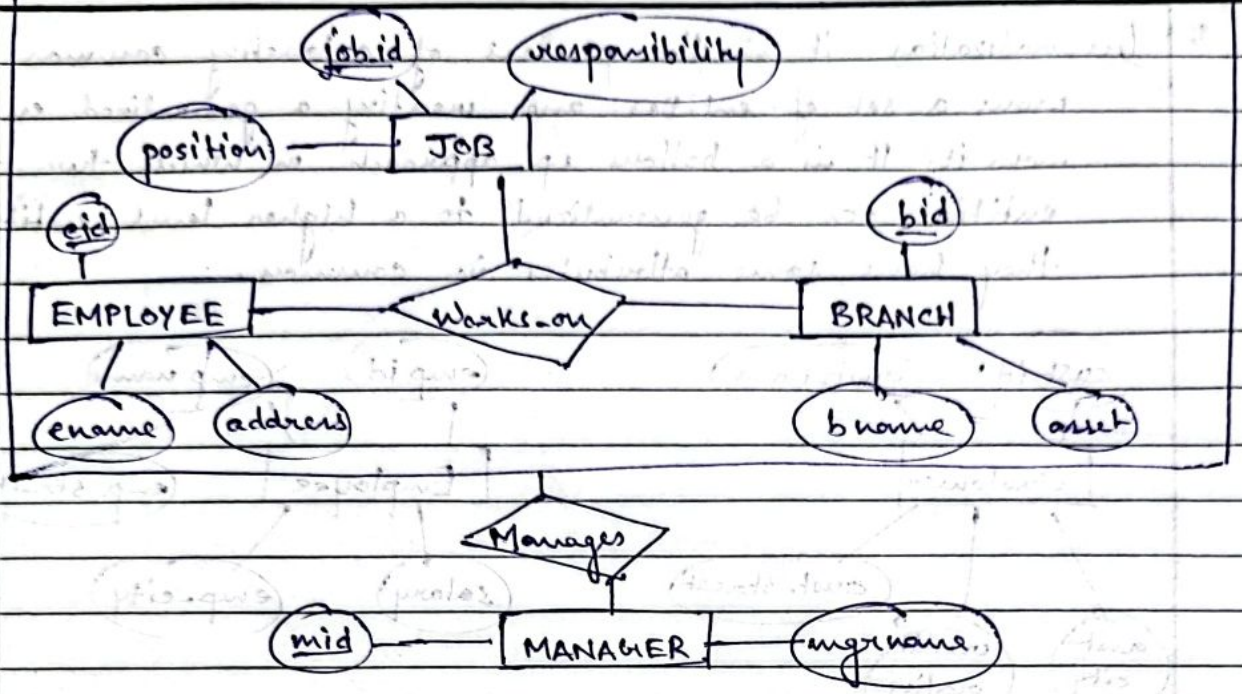
- Generalization It is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common.





- Aggregation An ER Model is not capable of representing the relationship b/w an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity.

  Aggregation is an ~~entity~~ abstraction through which we can represent relationships as higher level entity set.



Constraints on generalization

(a) Condition defined or not

- Condition defined - In this lower level entity sets, evaluation of memberships is on basis whether an entity satisfies an explicit condition or not. All the lower level entities are evaluated on the basis of the same attribute, thus it is also referred to as attribute defined.

- User defined - In this lower level entity sets are not constrained by a condition named membership; users of ~~enti~~ database assigns entities to a given entity set.

(b) Disjoint or overlapping

- Disjoint - A ~~disjointe~~ disjointness constraint requires that an entity belongs to only one lower level entity set.

- Overlapping - In overlapping generalizations, the same entity may belong to more than one lower level entity set within a single generalization.

(c) Completeness Constraint

- Total generalization / specialization - According to this constraint, each higher level entity must belong to a lower level entity set.

- Partial generalization / specialization - According to this constraint, some higher level entities may not belong to any lower entity set.

Relational Model

The relational model represents how data is stored in Relational databases. A relational database consists of a collection of tables, each of which is assigned a unique name. Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE and AGE shown in the table

Table STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

Attributes : Attributes are the properties that define an entity. eg. ROLL_NO, NAME, ADDRESS

Relation Schema : A relation schema defines the structure of the relation and represents the name of the relation with its attributes. EX.: STUDENT (ROLL_NO, NAME, ADDRESS PHONE AND AGE) is the relation schema for STUDENT. If a schema has more than 1 relation, its called as relation schema.

Tuple: Each row in the relation is known as tuple

Relation Instance: The set of tuples of a relation at a particular instance of time is called relation instance

Degree: The number of attributes in the relation is known as degree of the relation.

Cardinality: The number of tuples in a relation is known as Cardinality.

Relation Key: These are basically the keys that are used to identify the rows uniquely or also help in identifying tables.

- **Primary Key:** Is a set of attributes which uniquely identify the tuples in relation or table. The primary key is a minimal super key, so there is one and only one primary key in any relationship

- **Candidate Key:** A candidate key is a set of attributes which uniquely identify the tuples in relation or table. Any attribute can contain NULL value. There can be more than one candidate key.

- **Super Key:** Super key is an attribute that is used to uniquely identifies all attributes in a relation. All super keys can't be candidate keys but the reverse is true. In relation, a number of super keys is more than the number of candidate keys.

- Foreign Key : A foreign key is a column or a group of columns used to identify a row uniquely of a different table

Relational Database

A relational database is a database consisting of multiple relations or tables. The information about an enterprise is broken up into parts, with each relation storing one part of the information.

Constraints in relational model While designing the relational model, we define some conditions which must hold for data present in the database are called Constraints.

- Domain Constraints : All the values that appear in a column of a relation must be taken from the same domain. This constraint can be applied by specifying a particular data type to a column.

- Entity Integrity : The entity integrity rule is designed to assure that every relation has a primary key and that the data values for that primary key are all valid. Usually the primary key of each relation is the first column

  Entity Integrity ensures that every primary key attribute is NOT NULL. Primary key performs the unique identification function in a relational model.

- Referential Integrity: In relational data model, associations b/w tables are defined by using foreign keys. A referential integrity ~~constant~~ constraint is a rule that maintains consistency among the rows of two ~~columns~~ relations.

  The rule states that if there is a foreign key in one relation, either each foreign key must match a primary key value in other table or else foreign key must be NULL

A foreign key that references its own relation is known as recursive foreign key. The linking b/w the foreign key and primary key allows a set of relations to form an integrated database.

- Operational Constraints :- These are the constraints enforced in the database by the business rules or real world limitations.

DATABASE LANGUAGES

① DDL (Data Defination Language)
It is used to define the conceptual schema - The defination includes the information of all the entity sets and their associated attributes as well as the relationships b/w the entity sets.
CREATE, ALTER, DROP, RENAME & TRUNCATE

② DML (Data Manipulation Language)
It is used to manipulate data in the database. A query is a statement in the DML that requests the retrieval of data from the database
SELECT, INSERT, UPDATE & DELETE

③ DCL (Data Control Language)
It allows changing the permissions on database structures
GRANT & REVOKE

④ TCL (Transaction Control Language)
It allows permanently recording the changes made to the rows stored in a table or undoing such changes.
COMMIT, ROLLBACK & SAVEPOINT

Query Languages

Language in which user requests information from the database are :

- Procedural language
- Non procedural language

The categories of different languages are :

- SQL
- Relational Algebra
- Relational Calculus
  - Tuple Relational Calculus
  - Domain Relational Calculus

Relational Algebra

Relational Algebra is a procedural language for manipulating relations. Relational algebra operations manipulate relations. That is, these operations use one or two existing relations to create a new relation.

- Fundamental Operators
  - Unary : SELECT, PROJECT, RENAME
  - Binary : UNION, SET DIFFERENCE, CARTESIAN PRODUCT

- Secondary Operators
  - INTERSECTION, NATURAL JOIN, DIVISION, ASSIGNMENT

The relational schemas used for different operations are :

- Customer (cust_name, cust_street, cust_city)
  - used to store customer details
- Branch (branch_name, branch_city, assets)
  - used to store branch details
- Account (acc_no, branch_name, balance)
  - stores the account details
- Loan (loan_no, branch_name, amount)
  - stores the loan details

- Depositor (cust_name, acc_no)
  - stores the detail about customers account
- Borrower (cust_name, loan_no)
  - used to store the details about the customers loan.

- SELECT OPERATOR (σ)

SELECT Operation is used to create a relation from another relation by selecting only those tuples or rows from the original relation that satisfy a specified condition. It is denoted by sigma (σ) symbol. This predicate appears as a subscript to σ.

The general syntax of select operator is

$$\sigma_{<\text{selection-condition}>}(<\text{relation name}>)$$

Query: Find the details of the loans taken from 'Bhubaneshwar Main' branch.

$$\sigma_{\text{branch\_name} = \text{'BhubaneshwarMain'}}(\text{Loan})$$

- The operators used in selection predicate may be $=, \neq, <, \leq, >, \geq$.

- Different predicates can be combined into a larger predicate by using connectors like : AND ($\wedge$), OR ($\vee$), NOT ($\neg$)

- PROJECT OPERATOR (π)

PROJECT Operation can be thought of as eliminating unwanted columns. It eliminates the ~~unwanted~~ duplicate rows. It is denoted by pie (π) symbol. The attributes needed to be appeared in the resultant relation appear as a subscript to π

The general syntax of project operator is

$$\pi_{<\text{attribute-list}>}(<\text{relation name}>)$$

Composition of relational operators

Relational algebra operators can be expressed together into a relational algebra expression to answer complex queries.

Q Find the name of the customers who live in Bhubaneshwar -

Customer

cust-name	cust-street	cust-city
Rishi	India bnk	New Delhi
Sarthak	MG Road	Bangalore
Manas	Shastri Nagar	Bhubaneshwar
Ramesh	MG Road	Bhubaneshwar
Mahesh	Juhu	Mumbai

$\Pi_{cust\_name}(\sigma_{cust\_city = 'Bhubaneshwar'}(Customer))$

cust-name
Manas
Ramesh

RENAME Operator

The results of relational algebra expressions do not have a name that can be used to refer them. It is useful to be able to give them names; the rename operator is used for this purpose. It is denoted by rho (ρ) symbol.

The general syntax of rename operator is

$\rho_X(E)$

UNION Compatibility

To perform the set ~~intersection~~ operations such as UNION, DIFFERENCE and INTERSECTION, the relations need to be union compatible for the result to be a valid relation.

$R1(a_1, a_2, \ldots a_n)$ and $R2(b_1, b_2, \ldots b_m)$

(i) $n = m$ i.e. both relations have same arity

(ii) $dom(a_i) = dom(b_i)$ for $1 \le i \le n$

Page No:
Date:

UNION OPERATOR (∪)

The Union Operation is used to combine data from two relations. It is denoted by union (∪) symbol. The union of two relation $R_1(a_1, \dots a_n)$ and $R_2(b_1, b_2 \dots b_n)$ is a relation $R_3(c_1, c_2 \dots c_n)$ such as

$$dom(ci) = dom(ai) \cup dom(bi), \quad 1 \le i \le n$$

DIFFERENCE Operator (−)

The Difference Operation is used to identify the rows that are in one relation and not in another. It is denoted as (−) symbol. The difference of two relations $R_1(a_1, a_2, \dots a_n)$ and $R_2(b_1, b_2, \dots b_n)$ is a relation $R_3(c_1, c_2, \dots c_n)$ such that

$$dom(ci) = dom(ai) - dom(bi), \quad 1 \le i \le n$$

$R_1 - R_2$ is a relation that includes all tuples that are in R_1 but not in R_2.

Cartesian Product Operator (×)

The Cartesian Product of two relations $R_1(a_1, a_2 \dots a_n)$ with cardinality i and $R_2(b_1, b_2 \dots b_m)$ with cardinality j is a relation R_3 with

- degree $K = n + m$
- cardinality $i * j$
- attributes $(a_1, a_2, \dots a_n, b_1, b_2, \dots b_m)$

$R_1 \times R_2$ is a relation that includes all the possible combinations of tuples from R_1 and R_2.

Intersection Operator ($\cap$)

The intersection operator is used to identify the rows that are common to two relations. It is denoted by ($\cap$) symbol. The intersection of two relations $R_1(a_1, a_2, \ldots a_n)$ and $R_2(b_1, b_2, \ldots b_n)$ is a relation $R_3(c_1, c_2, \ldots c_n)$ such that

$$dom(c_i) = dom(a_i) \cap dom(b_i), \quad 1 \le i \le n$$

$R_1 \cap R_2$ is a relation that includes all tuples that are present in both R_1 and R_2

$$R \cap S = R - (R - S)$$

Join Operator ($\bowtie$)

The join is a binary operation that is used to combine certain selections and a cartesian product into one operation. It is denoted by $\bowtie$ symbol.

The join operation forms a cartesian product of its two arguments, performs a selection forcing ~~equity~~ equality on those attributes that appear in both relations and finally removes the duplicate attributes

Query: Find the names of customers who have a loan at the bank, along with the loan number and the loan amount

(i) $\Pi_{cust\_name, loan.loan\_no, amount}(\sigma_{Borrower.loan\_no = loan.loan\_no}(Borrower \times Loan))$

(ii) $\Pi_{cust\_name, loan\_no, amount}(Borrower \bowtie Loan)$

Page No:
Date:

Relational Database Design

Approaches of database design

- Bottom up design methodology
  - Design by synthesis
  - Relations created on the basis of relationships among individuals attributes

- Top down design methodology
  - Design by analysis
  - Relations created first are analyzed individually and collectively leading to further decompositions until all desired properties are satisfied.

Informal design guidelines

① Semantics of relation attributes
- Attributes must have real world meaning &
- must have proper interpretation to relations

② Reducing the redundant values in tuples
- Minimizes the storage space used by base relations

③ Reducing the NULL values in tuples
- If many attributes of a relation do not apply to all tuples of a relation, then we end up with many NULL values in those tuples.

Leads to
- Wastage of memory space
- Difficulty in understanding the meaning
- Problem for aggregate functions : SUM, AVG, COUNT etc.

④ Disallowing the possibility of generating spurious tuples

Page No:
Date:

Anomalies in Relational Model

An anomaly is an irregularity or something that deviates from expected or normal state. When designing databases, we identify three types of anomalies : i) Insert
ii) Update
iii) Delete

Table: Student

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

Table Branch

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGG.
CV	CIVIL ENGINEERING

Insertion Anomalies

We can't insert a row in REFERENCING RELATION if referencing attributes values isn't present in the referenced attribute value.

Ex: Insertion of a student with BRANCH-CODE 'ME' in Student relation will result in an error because 'ME' is not present in BRANCH_CODE of BRANCH

Deletion / Updation Anomaly in Referenced Relation

We can't delete or update a row from REFERCED RELATION if the value of REFERENCED ATTRIBUTE is used in the value of REFERENCING ATTRIBUTE

Ex: If we want to delete a tuple from BRANCH having BRANCH_CODE 'CS' it will result in an error because 'CS' is referenced by BRANCH_CODE of student but if we try to delete the row

from BRANCH with BRANCH_CODE CV, it will be deleted as the value is not been used by referencing relation

Advantages of Relational Model

- Simple to use in comparision to other languages
- More secure than any other relational model
- Data is more accurate in relational data model
- Data integrity is maintained

Disadvantages of Relational Model

- Not good for large databases
- Sometimes difficult to find relation b/w tables
- Because of complex structure, response time for queries is high.

Characteristics of Relational Model

- Data represented in data and colums called relations
- Data stored in tables having relationship b/w them is called relational model
- Supports operations like Data ~~Operations~~ definition, Data manipulation and transaction management
- Each column has distinct name and they are representing attributes
- Each row represents a single entity.

Page No:
Date:

Functional Dependencies

A functional dependency is a constraint b/w two sets of attributes from a relation

Let x and y are two sub sets of a relation schema R
A functional dependency $x \rightarrow y$ is holds on R if for any two tuples t_1 and t_2 in $r(R)$ that have

$$t_1[X] = t_2[X]$$

then they must also be in

$$t_1[Y] = t_2[Y]$$

Let r be the relation defined on relation schema R(A,B,C,D) as follows: $B \rightarrow D$

$t_1[B] = t_3[B]$ implies that $t_1[D] = t_3[D]$

R	A	B	C	D
t_1	a1	b1	c1	d1
t_2	a2	b2	c2	d2
t_3	a3	b3	c3	d3
t_4	a4	b4	c4	d4

Functional Dependency

Trivial Functional Dependency

- $A \rightarrow B$ has trivial dependency if B is a subset of A
- Following dependencies are also trivial
  - $A \rightarrow A$
  - $B \rightarrow B$

Non trivial functional dependency

- $A \rightarrow B$ has non trivial dependency if B is not a subset of A
- When A intersection B is NULL, then $A \rightarrow B$ is complete non trivial

Page No:
Date:

Armstrong's Axioms on Functional Dependency

This is the syntactic way of computing / testing the various properties of functional dependencies

① Reflexivity

If $Y \subseteq X$, then $X \rightarrow Y$ (trivial FD)

Name, Address → Name

② Augmentation

If $X \rightarrow Y$, then $XZ \rightarrow YZ$

If Town → Zip, then Town, Name → Zip, Name

③ Transitivity

If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

④ Union

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

⑤ Decomposition

If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

⑥ Pseudo transitivity

If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$

Proof of Armstrong's Axioms

① Reflexivity

If $Y \subseteq X$, then $X \rightarrow Y$ (trivial FD)

Since $Y \subseteq X$, there exists two tuples t_1 and t_2 in $r(R)$ such that

$$t_1[X] = t_2[X]$$

then

$$t_1[Y] = t_2[Y] \Rightarrow X \rightarrow Y$$

Page No:
Date:

② Augmentation

If $X \rightarrow Y$, then $XZ \rightarrow YZ$

Proof by contradiction: Assume that $X \rightarrow Y$, then $XZ \rightarrow YZ$

$X \rightarrow Y \Rightarrow t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$ — ①

$XZ \rightarrow YZ \Rightarrow t_1[XZ] = t_2[XZ]$ implies $t_1[YZ] = t_2[YZ]$ — ②

$t_1[XZ] = t_2[XZ]$ means $t_1[X] = t_2[X]$

and $t_1[Z] = t_2[Z]$ — ③

Equation ① and ③ implies $t_1[YZ] = t_2[YZ]$

contradicts eq. ②.

③ Transitivity

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

Proof

$X \rightarrow Y \Rightarrow t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$ — ①

$Y \rightarrow Z \Rightarrow t_1[Y] = t_2[Y]$ then $t_1[Z] = t_2[Z]$ — ②

From ① and ②,

$t_1[X] = t_2[X]$ then $t_1[Z] = t_2[Z] \Rightarrow X \rightarrow Z$

④ Union

If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

Proof:

$X \rightarrow Y$

$XX \rightarrow XY$ (augmenting X on both sides)

$X \rightarrow XY$ — ①

$X \rightarrow Z \Rightarrow YX \rightarrow YZ$ (Aug. rule) — ②

From ① and ②, implies $X \rightarrow YZ$

⑤ Decomposition

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

Proof: $Y \subseteq YZ$

$YZ \to Y$ (Reflexivity rule) — ①

$X \to YZ$ (Given) and Eq (1) $\Rightarrow X \to Y$ Proved

Similarly, $Z \subseteq YZ$

$YZ \to Z$ (Reflexivity rule) — ②

$X \to YZ$ (Given) and Eq (2) $\Rightarrow X \to Z$ Proved

⑥ Pseudo transitivity

If $X \to Y$ and $WY \to Z$, then $WX \to Z$

Proof: $X \to Y$ (Given)

$WX \to WY$ (Aug rule) — ①

Eq. ① and $WY \to Z$ (Given)

$WX \to Z$ Proved,

Entailment, Equivalence

If F is a set of functional dependencies on schema R and G is another FD on R, then F entails G, if every instance $r(R)$ that satisfies every FD in F also satisfies G

$F = \{A \to B, B \to C\}$ and $G = \{A \to C\}$ F entails G but G doesn't entails F.

$F = \{A \to B, A \to C\}$ and $H = \{A \to BC\}$ F entails H and H entails F

F equivalent to H

Equivalence, Closure

F and G are equivalent if F entails G and G entails F

The closure of F, denoted by F^+, is the set of all FDs entailed by F

Closure of a set of functional dependencies

Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F

Example

If $F = \{A \rightarrow B, B \rightarrow C\}$. Then using F we can infer that $A \rightarrow C$

So that $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

Let $R = \{A, B, C, G, H, I\}$ be a relation schema, with given set of functional dependencies F

$F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

Find the closure of F

Some members of F^+ includes

- $A \rightarrow H$ [By transitivity from $A \rightarrow B$ and $B \rightarrow H$]
- $AG \rightarrow I$ [By augmenting $A \rightarrow C$ with G to get $AG \rightarrow CG$ and then with transitivity from $CG \rightarrow I$]
- $CG \rightarrow HI$ [from $CG \rightarrow H$, $CG \rightarrow I$: Union rule
  Augmentation of $CG \rightarrow I$ to infer $CG \rightarrow CGI$
  augmentation of $CG \rightarrow H$ to infer $CGI \rightarrow HI$
  and then transitivity]

AXIOMS ARE SOUND

If an FD $G: X \rightarrow Y$ can be derieved from a set of FDs F using the axioms, then G holds in every relation that satisfies every FD in F

Example : Given $X \rightarrow Y$ and $X \rightarrow Z$, then

$X \rightarrow XY$	Augmentation by X
$YX \rightarrow YZ$	Augmentation by Y
$X \rightarrow YZ$	Transitivity

Thus $X \rightarrow YZ$ is satisfied in every relation where both $X \rightarrow Y$ and $X \rightarrow Z$ are satisfied

Page No:
Date:

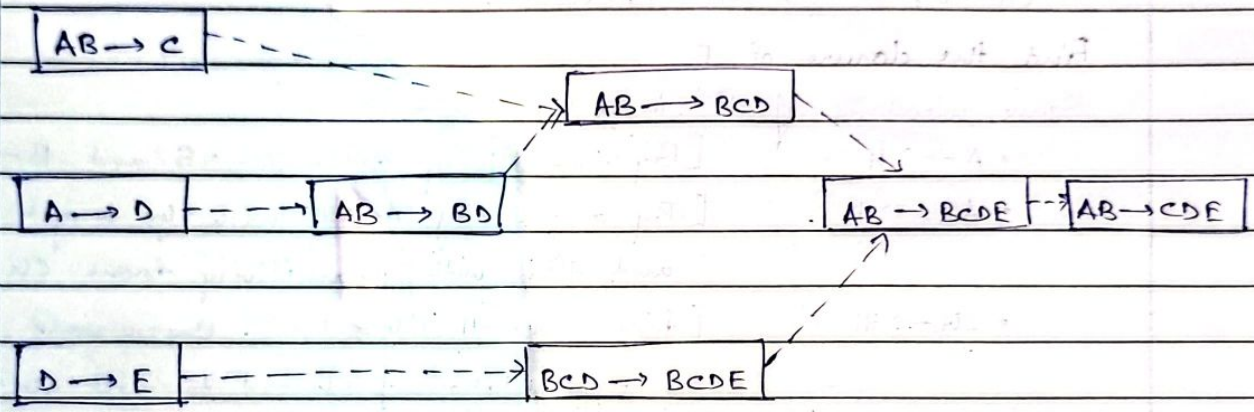
AXIOMS ARE COMPLETE

If F entails G_i, then G_i can be derived from F using the axioms

* A consequence of completeness in the following (naive) algorithms to determine if F entails G_i

Generating F^+

$F = \{AB \rightarrow C, A \rightarrow D, D \rightarrow E\}$



Attribute Closure

The attribute closure of a set of attributes, X with respect to a set of functional dependencies, F (denoted by X_F^+) is the set of all attributes, A such that $X \rightarrow A$

X_{F1}^+ is not necessarily same as X_{F2}^+ for $F_1 \neq F_2$

Algo

Given a set of FDs, F, then $X \rightarrow Y$ iff $X_F^+ \supseteq Y$

Page No:
Date:

Algorithm: Determining X^+, the closure of X under F

Input: Let F be a set of FDs for relation R

Steps:

1. $X^+ = X$ // initialize X^+ to X
2. For each FD: $Y \rightarrow Z$ in F Do
   If $Y \subseteq X^+$ then // if Y contained in X^+
   $X^+ = X^+ \cup Z$ // add Z to X^+
   End if
   End for
3. Return X^+ // return closure of X

Output: Closure X^+ of X under F

* If $T \in X^+$, then $X \rightarrow T$ is entailed by F

Q. $F: [AB \rightarrow C, A \rightarrow D, D \rightarrow E, AC \rightarrow B]$

Compute the attribute closure of
i) A ii) AB iii) B iv) D

X	X_F^+
A	[A, D, E]
AB	[A, B, C, D, E]
B	[B]
D	[D, E]

Uses of attribute closure

① Testing for super key
To test α is a super key, we compute α_F^+ and check if α_F^+ contains all attributes of R

② Testing functional dependencies
To check if a functional dependency $\alpha \rightarrow \beta$ holds (or in other words is in F^+) just check if $\beta \subseteq \alpha_F^+$

③ Computing closure
For each $\gamma \subseteq R$, we find the closure of γ_F^+ and for each $S \subseteq \gamma_F^+$ we output a functional dependency $\gamma \rightarrow S$

Minimal (Canonical) Cover of a set of dependencies, F

① Make RHS of each FD into single attribute of F
② Eliminate redundant attributes from LHS of each FD of F
③ Delete redundant FDs from given set of FDs of F

A minimal cover of a set of dependencies T is a set of dependencies U such that

- U is equivalent to T $(T^+ = U^+)$
- All FDs in U have the form $X \rightarrow A$ where A is a single attribute
- It is not possible to make U smaller by
  - Deleting an FD
  - Deleting an attribute from an FD
- FDs and attributes that can be deleted in this way are called redundant

Q. Find the minimal cover of

$T = \{ ABH \rightarrow CK, A \rightarrow D, C \rightarrow E, BGH \rightarrow F, F \rightarrow AD, E \rightarrow F, BH \rightarrow E \}$

$F \rightarrow AD$ replaced by $F \rightarrow A, F \rightarrow D$

$ABH \rightarrow CK$ replaced by $ABH \rightarrow C, ABH \rightarrow K$

$T = \{ ABH \rightarrow C, ABH \rightarrow K, A \rightarrow D, C \rightarrow E, BGH \rightarrow F, F \rightarrow A, F \rightarrow D, E \rightarrow F, BH \rightarrow E \}$

$T = \{ BH \rightarrow C, BH \rightarrow K, A \rightarrow D, C \rightarrow E, BGH \rightarrow F, F \rightarrow A, F \rightarrow D, E \rightarrow F, BH \rightarrow E \}$

$T = \{ BH \rightarrow C, BH \rightarrow K, A \rightarrow D, C \rightarrow E, F \rightarrow A, E \rightarrow F \}$

Page No:
Date:

Find the minimal cover for the following functional dependencies

$F = \{A \to B, C \to B, D \to ABC, AC \to D\}$

(i) $A \to B, C \to B, D \to A, D \to B, D \to C, AC \to D$

(ii) $A \to B, C \to B, D \to A, D \to C, AC \to D$

(iii) $A \to B, C \to B, D \to A, D \to C, AC \to D$

Finding Candidate Key

In a relational model, the set of those attributes which can be used to identify each row uniquely is called super key and there can be a lot of super keys. Those super keys which have no proper subset as any super key, then it's called a candidate key

For example, in relational model, we have three attributes as $\{A,B,C\}$ and $\{AB\}$ can identify a complete table and $\{A\}$ itself can identify the whole table then AB and A both will be super key but A will be the candidate key.

Steps to find out candidate key:

① We will find out the essential and non essential sets of attributes from a given set of attributes. Those attributes which are dependent on other attributes are non essential attributes and their values can be found out using essential attributes. So the essential attributes will definitely be a part of candidate key.

② We will combine all the essential attributes and if they can determine the attributes (by finding their closure) then it will be the candidate key.

③ If a combination of essential attributes is not a candidate key then we will try different combinations of essential and non essential attributes which are not subset of each other to find out all the candidate keys.

* Attributes which never present on RHS of any functional dependencies are called as essential attributes

Page No:
Date:

Q Func. dependencies = {A→B, B→C, C→A}
or S = {A, B, C, D}

We have non essential attributes as {A, B, C} and essential attributes as {D}

So D will be the part of candidate key

Closure of D (D+) = {D}

Now,

AD+ = {A, B, C, D} Using A→B and B→C

So AD will be the candidate key

BD+ = {A, B, C, D} Using C→A and B→C

So BD will be the candidate key

CD+ = {A, B, C, D} Using C→A and A→B

So CD will be the candidate key

Candidate Keys are : {AB, BD, CD}

Q S: {A, B, C, D, E, F} functional dependencies are : AB→C
B→AE
C→D

Non essential attributes are : {A, C, D, E}

Essential attributes are : {B, F}

BF will be the part of candidate key

Closure of BF (BF+) = {B, A, E, C, D, F}

Since combination of essential attributes is capable of finding all the attributes, it will be a candidate key

All other combinations of essential and non essential will not be minimal.

So there will be only one candidate key as {BF}

Page No:
Date:

Q $S = \{W, X, Y, Z\}$ functional dependencies are: $Z \to W$, $Y \to XZ$, $XW \to Y$

Non essential attributes : $\{W, X, Y, Z\}$

Essential attributes : $\{\}$.

$W^+ = \{W\}$

$X^+ = \{X\}$

$Y^+ = \{Y, X, Z, W\}$ // Using $Y \to XZ$ and $Z \to W$

$Z^+ = \{Z\}$

Y is capable of finding all the attributes so it will be a candidate key. Y will not be a part of next combinations.

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

$ZW^+ = \{Z, W, \}$

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

So XW and ZX will be the next candidate keys and the next other combinations will contain no attributes apart from these candidate keys

Candidate Keys are: $\{Y, XW, ZX\}$

No. of Super Keys $= 2^{n - (\text{size of candidate key})}$

where n is the no. of attributes

Relation consists of n attributes with each attribute is candidate key

$\Rightarrow$ No. of possible super keys $= 2^n - 1$

Relation consists of n attributes with two candidate keys (A_1 & A_2)

$\Rightarrow$ No. of possible super keys $= 2^{n-1} + 2^{n-1} + 2^{n-2}$

$|A_1| + |A_2| - |A_1 \cap A_2|$

$=$ (possible super keys with A_1) + (possible super keys with A_2) - (common super keys with A_1 and A_2)

Page No:
Date:

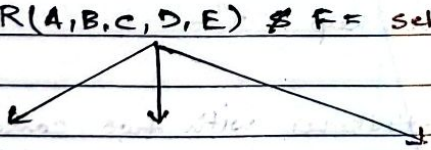
Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:
- Making relations very large
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources
- The likelihood of errors and inconsistencies increases.

To handle these problems, we should analyse and decompose the relations with ~~dc~~ redundant data into smaller, simpler and well structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes

- What is a Normal Form?

A Normal form is a state of a ~~design~~ relation that results from applying simple rules regarding functional dependencies (or relationship b/w attributes) to that relations, to achieve the following desired properties
- Minimizing redundancy
- Minimizing the insertion, deletion and modification anomalies



Suppose R is suffering from redundancy & different anomalies

* Data modification anomalies can be categorized into three types:

(i) Insertion Anomaly: It refers to when one cannot insert a new tuple into a relationship due to lack of data.

** Reason behind these anomalies is data redundancy

Page No:
Date:

(II) Deletion Anomaly: The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.

(III) Updatation Anomaly: The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Properties of decomposition during Normalization

(i) The lossless join or non-additive join property
(ii) The dependency preserving property
(iii) Attribute preserving property

Steps in Normalization

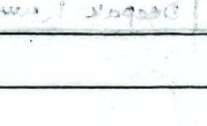
Relation State	Action
1 NF	Remove multivalue attributes
2 NF	Remove partial functional dependencies (pfd)
3 NF	Remove transitive functional dependencies (tfd)
BCNF	Remove other functional dependencies which are responsible for redundancy
4NF	Remove multivalue dependencies
5NF	Remove join dependencies

Lossless Join Decomposition

A decomposition of R into R into R_1 and R_2 is lossless join iff one of the following dependencies is in F*

$R_1 \cap R_2 \rightarrow R_1$

$R_1 \cap R_2 \rightarrow R_2$



$R = R_1 \cup R_2$

$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$

Page No:
Date:

Dependency preserving decomposition

Given R and F

Decompose F into $F_1, F_2, \ldots$ ... F_n so that each F_i be the set of dependencies F^+ that include only attributes in R_i

F_i is restriction of F to R_i

$F' = F_1 \cup F_2 \cup \ldots \cup F_n$

In general $F' \neq F$

But if $F'^+ = F^+$, then F and F^+ are equivalent

First Normal Form (1NF)

- The intersection of each row and each column of the table must contain a either a single value or possibly a NULL
- That is, any multivalued attributes (also called repeating groups) if present must be removed

EMPLOYEE

EmpID	EmpName	Skill
E1001	Ashok Kumar	{C, C++, Oracle}
E1002	Bijaya Kumar	{C++, Oracle}
E1003	Deepak Kumar	{C, C++}

Solutions

① EMPLOYEE

EmpID	EmpName	Skill 1	Skill 2	Skill 3
E1001	Ashok Kumar	C	C++	Oracle
E1002	Bijaya Kumar	C++	Oracle	NULL
E1003	Deepak Kumar	C	C++	NULL

(2) EMPLOYEE

EmpID	EmpName	Skill
E1001	Ashok Kumar	C
E1001	Ashok Kumar	C++
E1001	Ashok Kumar	Oracle
E1002	Bijaya Kumar	C++
E1002	Bijaya Kumar	Oracle
E1003	Deepak Kumar	C
E1003	Deepak Kumar	C++

(3) EMPLOYEE

EmpID	EmpName
E1001	Ashok Kumar
E1002	Bijaya Kumar
E1003	Deepak Kumar

EmpID	Skill
E1001	C
E1001	C++
E1001	Oracle
E1002	C++
E1002	Oracle
E1003	C
E1003	C++

Second Normal Form (2NF)

A relation is in 2NF if

- It is in 1NF
- Every non key attributes is fully functional dependent on the primary key (i.e should ~~have~~ not have any partial functional dependency - pdf)

$R(A,B,C,D)$ $F = (AB \rightarrow CD, B \rightarrow D)$

- Here AB is the Key as $AB \rightarrow CD$
- The fd $(B \rightarrow D)$ is a pfd as a key attribute (i.e, prime attribute 'B' determines a non key attribute (non prime attribute 'D'))

Decompose R $R(A,B,C,D)$ $(AB \rightarrow CD, B \rightarrow D)$

$R1(B,D)$ $R2(A,B,C)$

$(B \rightarrow D)$ $(AB \rightarrow C)$

Page No:
Date:

Q. R = EMPLOYEE (EmpID, CourseTitle, EmpName, DeptName, Salary, CompletionDate)
and a set of functional dependencies
F = { EmpID, CourseTitle → CompletionDate ; EmpID → EmpName, DeptName, Salary }

EMPLOYEE (EmpID, CourseTitle, EmpName, DeptName, Salary, CompletiaDate)

EMPL (EmpID, EmpName, DeptName, Salary)
{ EmpID → EmpName, DeptName, Salary }

EMPLCourse (EmpID, CourseTitle, CompletiaDate)
{ EmpID, CourseTitle → CompletionDate }

THIRD NORMAL FORM (3NF)

A relation is in third normal form (3NF) if
- It is in 2NF
- No transitive functional dependencies exist

* A transitive dependency is a dependency among non key attributes
All 3NF decompositions are lossless join and dependency preserving decompositions

R(A,B,C,D) and F = { A → BCD, C → D }
- Here A is the key as A → BCD
- The FD { C → D } is a tfd as a non key attribute (i.e. non prime attribute, C) determines another non key attribute (non prime attribute, D)

R (A,B,C,D) { A → BCD, C → D }

R1 (C,D) { C → D }

R2 (A,B,C) { A → BC }

Page No:
Date:

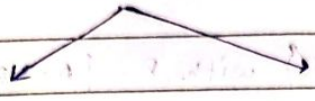
Given relation schema

SALES (CustID, CustName, SalesPoint, Region)

and set of functional dependencies

F = {CustID → CustName, SalesPoint, Region ; SalesPoint → Region}

SALES (CustID, CustName, SalesPoint, Region)



SALES1 (SalesPoint, Region)
{SalesPoint → Region}

SALES2 (CustID, CustName, SalesPoint)
{CustID → CustName, SalesPoint}

A relation is in third normal form if it holds atleast one of the following conditions for every non trivial functional dependency $X \rightarrow Y$

1. X is a super key
2. Y is a prime attribute i.e. each element of Y is part of some candidate key

- If A → B and B → C are two FDs then A → C is called transitive dependency

Stud_No	Stud_Name	Stud_State	Stud_Country	Stud_Age
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

FD set: {Stud_No → Stud_Name, Stud_No → Stud_State, Stud_State → Stud_Country, Stud_No → Stud_Age}

Candidate Key: {Stud_No}

For this relation, Stud_No → Stud_State and Stud_State → Stud_Country are true. So Stud_Country is transitively dependent on Stud_No. It violates the third normal form. To convert it into the third normal form we decompose the relation STUDENT (Stud_No, Stud_Name, Stud_Phone, Stud_State, Stud_Country, Stud_Age) as

Page No:
Date:

STUDENT (Stud.No, Stud-Name, Stud.Phone, Stud-State, Stud-Age)
STATE-COUNTRY (STATE, COUNTRY)

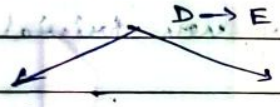
Given relation schema
R(A,B,C,D,E,F) and set of functional dependencies
F = {A → BCDEF, BC → ADEF, B → F, D → E}

R(A,B,C,D,E,F) with F = {A → BCDEF, BC → ADEF, B → F, D → E}

R1(B,F)
F = {B → F}

R2(A,B,C,D,E)
F = {A → BCDE, BC → ADE, D → E}

$A_F^+ = R$
$(BC)_F^+ = R$
$(B)_F^+ = BF$
$(D)_F^+ \subset DE$



R21(D,E)
F = {D → E}

R22(A,B,C,D)
F = {A → BCD, BC → AD}

In D → E, neither D is super key nor E is a prime attribute

Here all the relations appearing as a leaf node are in 3NF. All 3NF decompositions are lossless, non additive, and dependency preserving decompositions

BCNF (Boyce Codd Normal Form)

- BCNF is the advanced version of 3NF, it is stricter than 3NF
- A table is in BCNF if every functional dependency $X \rightarrow Y$, X is the super key of the table
- For BCNF the table should be in 3NF and for every FD, LHS is a Super Key

Page No:
Date:

EMPLOYEE

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	INDIA	DESIGNING	D394	283
264	INDIA	TESTING	D394	300
364	UK	STORES	D283	232
364	UK	DEVELOPING	D283	549

FDs

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate Key : {EMP_ID, EMP_DEPT}

* The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys

① EMP_COUNTRY table

EMP_ID	EMP_COUNTRY
264	INDIA
364	UK

② EMP_DEPT table

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
DESIGNING	D394	283
TESTING	D394	300
STORES	D283	232
DEVELOPING	D283	549

③ EMP_DEPT_MAPPING

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

FDs

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate Keys

For 1st table : EMP_ID

For 2nd table : EMP_DEPT

For 3rd table : {EMP_ID, EMP_DEPT}

Page No:
Date:

Q Given relation schema:

R(A,B,C,D,E,F) and a set of functional dependencies

F = {A → BCDEF, BC → ADEF, B → F, D → E}

Is R in BCNF ?

R(A,B,C,D,E,F) with F = {A → BCDEF, BC → ADEF, B → F, D → E}

In B → F, B is not a superkey

In D → E, D is not a superkey

R1(B,F) R2(D,E) R3(A,B,C,D)

$A_F^+ = R$, $(BC)_F^+ = R$

$B_F^+ = BF$, $D_F^+ = DE$

All FDs are non trivial

Lossless (non additive) join test for n-ary decomposition

Example 1

R = {SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS}

F = {SSN → ENAME ; PNUMBER → (PNAME, PLOCATION); (SSN, PNUMBER) → HOURS}

R1 = {ENAME, PLOCATION}

R2 = {SSN, PNUMBER, PNAME, PLOCATION, HOURS}

	1	2	3	4	5	6
	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS
R1	b11	a2	b13	b14	a5	b16
R2	a1	b22	a3	a4	a5	a6

Test → Fail

After examining all fds, No change found. That is No Row has all aj

Example 2

R = (SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS)

F = {SSN → ENAME ; PNUMBER → (PNAME, PLOCATION); (SSN, PNUMBER) → HOURS)

R1 = EMP = (SSN, ENAME)

R2 = PROJ = (PNUMBER, PNAME, PLOCATION)

R3 = WORKS_ON (SSN, PNUMBER, HOURS)

.	1	2	3	4	5	6	
	SSN	ENAME	PNUMBER	PNAME	PLOCATION	HOURS	
R1	a1	a2	b13	b14	b15	b16	test-fail
R2	b21	b22	a3	a4	a5	b26	
R3	a1	b32	a3	b34	b35	a6	

⊛ Example 3

R = { SSN, ENAME, PNUMBER, PNAME, PLOCATION, HOURS }

F = { SSN → ENAME ; PNUMBER → (PNAME, PLOCATION) ; (SSN, PNUMBER) → HOURS }

Page No:
Date:

Multivalued Dependencies

A multivalued dependency (MVD) $X \twoheadrightarrow Y$ specified on relation schema R where X and Y are both subsets of R specifies the following constraint on any relation state r of R:

If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in r with the following properties, where we use Z to denote $(R - (X \cup Y))$:

$t_3[X] = t_4[X] = t_1[X] = t_2[X]$

$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$

$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$

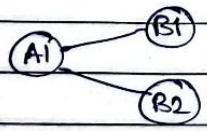
X	Y	Z	tuple
Kumar	Database	Rakesh	t_1
Kumar	Testing	Rao	t_2
Kumar	Database	Rao	t_3
Kumar	Testing	Rakesh	t_4

An MVD $X \twoheadrightarrow Y$ in R is called a trivial MVD if

(a) Y is a subset of X OR

$X \cup Y = R$

- A multivalued dependency $A \twoheadrightarrow B$ exists iff for every occurance of A, there exists multiple occurances of B



- If $A \twoheadrightarrow B$ and $A \twoheadrightarrow C$, then we have a single attribute A which multi determines two other attributes B and C

- Multi valued dependencies are also referred to as tuple generating dependencies

A	B
A1	B1
	B2

→

A	B
A1	B1
A1	B2

Page No:
Date:

4th NORMAL FORM (4NF)

A relation is in 4NF iff the following two conditions are satisfied simultaneously:

(a) If it is in BNF

(b) It contains no multiple MVDs

OR

A relation schema R is in 4NF wrt to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every non trivial multivalued dependency $X \twoheadrightarrow Y$ in F^+, X is a super key for R

Q. The EMP relation with two MVDs

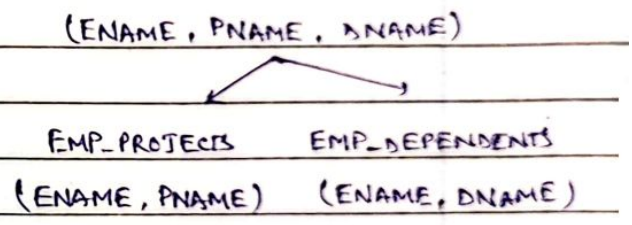
i. ENAME $\twoheadrightarrow$ PNAME

ii. ENAME $\twoheadrightarrow$ DNAME

(b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS

EMP

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John



EMP_PROJECTS

ENAME	PNAME
Smith	X
Smith	Y

EMP_DEPENDENTS

ENAME	DNAME
Smith	John
Smith	Anna

UNIT - 4

TRANSACTION PROCESSING AND ERROR RECOVERY

Transactions : A unit of program execution that accesses and possibly updates some data items

- A transaction is a collection of operations that logically form a single unit.
- Executing a transaction : either all operations are executed or none are
- Each transactions may consists of several steps, some involving I/O activity and some CPU activity.
- Moreover typically several transactions are running on a system : some are long some are short.

- Transaction Model
  - Operation read (X)

    Transfers the data item X from the database to a local buffer belonging to a transaction.

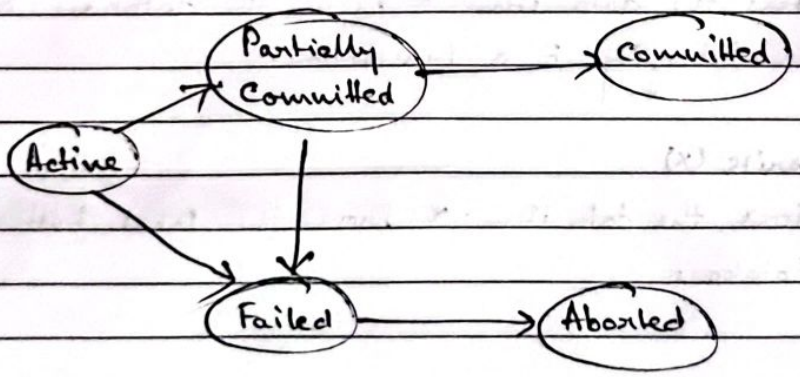
  - Operation write (X)

    Transfers the data item X from the local buffer back to the database.

- ACID Properties
  - Atomicity : Either all operations of a transactions are reflected properly in the database, or none are
  - Consistency : Execution of a transaction in isolation preserves the consistency of the database.
  - Isolation : Even though many transactions may run concurrently, the DBMS ensures that for any two transactions T, T' it appears to T that either T' finished before T started or T' started execution after T finished.
  - Durability : After a transaction completes successfully, the changes it has made persist

- States of a transaction

- Active : It stays in this state while executing
- Partially committed : After the final statement has been executed
- Failed : After the discovery that normal execution cannot proceed
- Aborted : After it has been rolled back and the database state restored to the one prior to the start of the execution.
- Commited : After successful completion



Two transactions

- T takes $100 from account A to account B
- T' takes 10% of account A to account B
- Property of T and T' : they don't change A+B
  Money isn't created and doesn't disappear

Page No:
Date:

T:	T':
read (A);	read (A);
A := A-100;	tmp := A * 0.1;
write (A);	A := A - tmp;
read (B);	write (A);
B = B + 100;	read (B);
write (B)	B = B + tmp;
	write (B)

Two serial executions : T ; T' and T', T

T	T'	T'	T
read (A);		read (A);	
A := A - 100;		tmp := A * 0.1;	
write (A);		A := A - tmp;	
read (B);		write (A);	
B = B + 100;		read (B);	
write (B);		B = B + tmp;	
	read (A);	write (B);	
	tmp := A * 0.1;		read (A);
	A := A - tmp;		A := A - 100;
	write (A);		write (A);
	read (B);		read (B);
	B = B + tmp;		B = B + 100;
	write (B);		write (B);

Transaction invariant

- Assume that both A and B have $1000
- Evaluating T ; T';
  - after T: A = 900, B = 1100
  - after T': A = 810, B = 1190
  - A + B = 2000
- Evaluating T'; T;
  - after T': A = 900, B = 1100
  - after T: A = 800, B = 1200
  - A + B = 2000

Page No:
Date:

Concurrent Execution :

T	T'
read (A);	
A := A - 100	
write (A);	
	read (A);
	tmp := A * 0.1;
	A := A - tmp;
	write (A);
read (B);	
B = B + 100;	
write (B);	
	read (B);
	B = B + tmp;
	write (B);

Result :
A = 810
B = 1190
A + B = 2000

WR
RW
WW
RR

Concurrent execution II

T	T'
read (A);	
A := A - 100;	
	read (A);
	tmp := A * 0.1
	A := A - tmp
	write (A);
write (A);	
read (B);	
B = B + 100	
write (B)	
	read (B);
	B = B + tmp
	write (B)

Result
A = 900
B = 1200
A + B = 2100
We created $100

Note : A concurrent schedule is serializable if it is equivalent to a serial schedule

Serializability

- Why is schedule I good and schedule II bad ?
  Because schedule I is equivalent to a serial execution of T and T' and schedule II is not.
- We formalize this via CONFLICT SERIALIZABILITY
- Transaction scheduling in DBMS always ensures serializability

SIMPLIFIED REPRESENTATION OF TRANSACTIONS

- For scheduling, the only important operations are read & write. What operations are performed on each data item doesn't affect the schedule.
- We thus represent transactions by a sequence of read - write operations assuming that between each read(A) and write(A) some computation is done on A

Examples of two concurrent executions in the new model

T	T'
read(A);	
write(A);	
	read(A);
	write(A);
read(B);	
write(B);	
	read(B);
	write(B);

T	T'
read(A);	
	read(A);
	write(A);
write(A);	
read(B);	
write(B);	
	read(B);
	write(B);

Analysing Conflicts

- Let op1 and op2 be two consecutive operations in a schedule
- Conflict - the order matters

  op1 ; op2 and op2 ; op1

  may give us different results
- If there is no conflict, op1 and op2 can be swapped
- If op1 and op2 refer to different data items, they do not cause a conflict and can be swapped

- If they are both operations on the same data items, they do not cause a conflict and can be swapped.

- If they are both operations on the same data item x, then
  if both are read(X), the order doesn't matter
  if op1 = read(X), op2 = write(X), the order matters
  if op1 = write(X), op2 = read(X), the order matters
  if op1 = write(X), op2 = write(X), the order matters

- Swapping a pair of operations in a schedule is allowed then,
  they refer to different data items or
  they refer to same data item and are both read operations.

- A schedule is called Conflict Serializable if it can be transformed into a serial schedule by a sequence of such swap operations

Schedule 1 is conflict serializable

T	T'	T	T'	T	T'
read(A)		read(A)		read(A)	
write(A)		write(A)		write(A)	
	read(A)		read(A)	read(B)	
	write(A) →	read(B)	→		read(A)
read(B)			write(A)		write(A)
write(B)		write(B)		write(B)	
	read(B)		read(B)		read(B)
	write(B)		write(B)		write(B)

Page No:
Date:

T	T'	T	T'
read(A)		read(A)	
write(A)		write(A)	
read(B)		read(B)	
	read(A) →	write(B)	
write(B)			read(A)
	write(A)		write(A)
	read(B)		read(B)
	write(B)		write(B)

Schedule II is not conflict serializable

T	T'
read(A)	
	read(A)
	write(A)
write(A)	
read(B)	
write(B)	
	read(B)
	write(B)

In a serial schedule, either:

write(A) by T precedes read(A) by T' (OR)

write(A) by T' precedes read(A) by T

But,

write(A) by T cannot be swapped with write(A) by T' (AND)

write(A) by T' cannot be swapped with read(A) by T

Hence the schedule is not serializable

Page No:
Date:

Testing for serializability

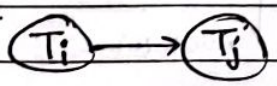
Serialization Graph is used to test the Serializability of a schedule

Assume a schedule S. For S, we construct a graph known as precedence graph. The graph has a pair $G = (V, E)$, where V consists of a set of vertices and E consists of the set of edges. The set of vertices is used to contain all the instructions participating in the ~~instruction~~ schedule. The set of edges is used to contain all edges $T_i \rightarrow T_j$ for which one of the three conditions holds:

(i) Create a node $T_i \rightarrow T_j$ if T_i executes write(Q) before T_j executes read(Q).

(ii) Create a node $T_i \rightarrow T_j$ if T_i executes read(Q) before T_j executes write(Q)

(iii) Create a node $T_i \rightarrow T_j$ if T_i executes write(Q) before T_j executes write(Q)

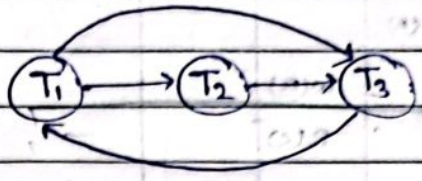


- If a precedence graph contains a single edge $T_i \rightarrow T_j$ then all instructions of T_i are executed before the first instruction of T_j is executed.

- If a precedence graph for schedule S contains a cycle then S is non serializable. If the precedence graph has no cycle then S is serializable.

- R1(X); R3(X); W1(X); R2(X); W3(X)

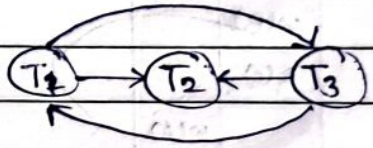
T_1	T_2	T_3
R1(X)		
		R3(X)
W1(X)		
	R2(X)	
		W3(X)



CYCLE ⇒ NOT Conflict Serializable

- R1(X); R3(X); W3(X); W1(X); R2(X)

T_1	T_2	T_3
R1(X)		
		R3(X)
		W3(X)
W1(X)		
	R2(X)	



$\forall A, B$

ω

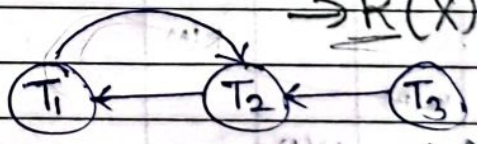
$\rightarrow R(A) = \omega(A)$

- R1(A), R2(A), R1(B), R2(B), R3(B), W1(A), W2(B)

T_1	T_2	T_3
R1(A)		
	R2(A)	
R1(B)		
	R2(B)	
		R3(B)
W1(A)		
	W2(B)	

$\forall X$

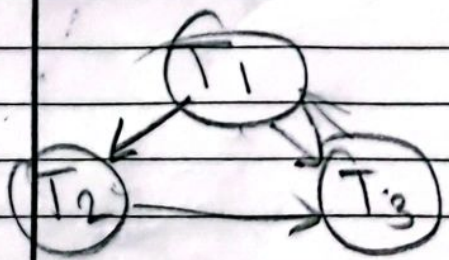
$\rightarrow R(X) = W(X)$



$W(X) = R(X)$

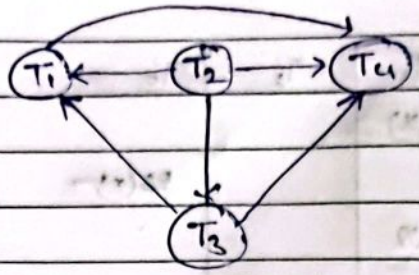
$W(X) = W(X)$

T_1	T_2	T_3
R1(X)		
		R3(X)
W1(X)		
	R2(X)	
		W3(X)



Page No:
Date:

T_1	T_2	T_3	T_4
	R(A)		
		W(A)	
W(A)			
	W(B)		
	R(C)		
			R(A)
			R(B)

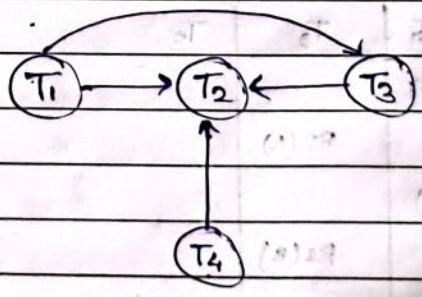


The Order of Serializability

$T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_4$

T_1	T_2	T_3	T_4
	R(A)		
	W(B)		
	R(C)		
		W(A)	
W(A)			
			R(A)
			R(B)

T_1	T_2	T_3	T_4
			R(A)
	R(A)		
		R(A)	
W(B)			
	W(A)		
		R(B)	
	W(B)		



The Order of serializability

$T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$

T_1	T_2	T_3	T_4
			R(A)
W(B)			
		R(A)	
		R(B)	
	R(A)		
	W(A)		
	W(B)		

Page No:
Date:

Locking based protocols for concurrency control

- Locking is a procedure used to control concurrent access to data.
- Lock enables a multi level database system to maintain the integrity of transactions by isolating a transaction from others executing concurrently.

Locking is one of the most widely used mechanisms to ensure serializability.

- How to ensure serializability ?

Access to data items must be done in mutually exclusive manner. That is when one transaction is accessing data item, then no other ~~item~~ transaction can modify that data item.

A Lock is a mechanism to control concurrency access to a data item.

{ A transaction must obtain a lock on a data item before it performs any read/write operation }

Types of locks:

① Exclusive lock (Write lock)

Lock-X(Q)

If a transaction Ti has obtained an exclusive mode lock(X) on data item Q, then Ti can both read and write Q.

T_1	T_1
Read(Q)	Lock-X(Q)
Read := Q * 1.10 →	Read(Q)
Write(Q)	Q := Q * 1.10
	Write(Q)

② Shared lock (Read Lock)

Lock-S(Q)

If a transaction Ti has obtained a shared mode lock(S) on data item Q, then Ti can only read data item Q but not write on Q

T_1	T_1
Read(P)	Lock-X(Q)
Read(Q)	Lock-S(P)
Q := Q + P →	Read(P)
Write(Q)	Read(Q)
	Q := Q + P
	Write(Q)
	Unlock(P)
	Unlock(Q)

③ Unlock(Q)

A transaction can unlock a data item Q

Lock Requests are made to concurrency control manager

Page No:
Date:

Lock Compatibility Matrix

	S	X
S	✓	X
X	X	X

- If a transaction has a shared / read lock on a data item, it can only read the item's but not update its value.
- If a transaction has a shared / read lock on a data item, other transactions can obtain read locks on the same data item, but they cannot obtain any update lock on it.
- If a transaction has a exclusive / write lock on a data item, then it can both read and update the value of that data item.
- If a transaction has a exclusive / write lock on a data item, then other transactions cannot obtain either a read lock or a write lock on that data item.

A locking protocol is a set of rules followed by all transactions while requesting and releasing locks.

Locking protocols restrict the set of possible schedules.

DEADLOCK

In the schedule, neither T_3 nor T_4 can make progress - executing lock-S(B) causes T_4 to wait for T_3 to release its lock on B, while executing lock-X(A) causes T_3 to wait for T_4 to release its lock on A.

SUCH A SITUATION IS CALLED DEADLOCK

T_3	T_4
lock-X(B)	
Read(B)	
B := B - 50	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.

{ The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil. }

Starvation

Starvation is also possible if concurrency control manager is badly designed. For example:

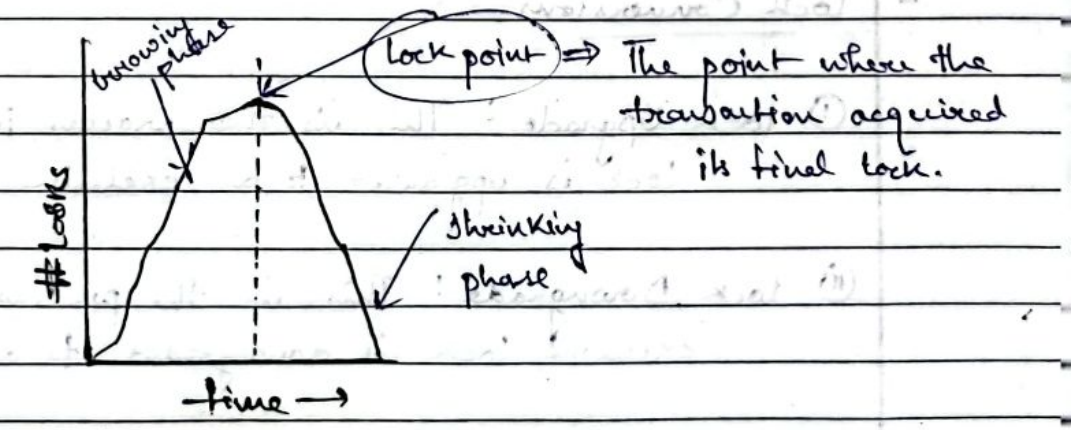
- A transaction may be waiting for a lock K on an item, while a sequence of other transactions request and are granted an lock (S) on the same item.
- The same transaction is repeatedly rolled out due to transactias.

{ Concurrency control manager can be designed to prevent starvation. }

Two Phase Locking Protocol

Phase 1: Growing Phase
→ locks are acquired and no locks are released

Phase 2: Shrinking Phase
→ locks are released and no locks are acquired.



- Two transactions cannot have conflicting locks
- This protocol ensures conflict serializable schedules
- Transactions can be serialized (ordered) in the order of their lock points.

— Limitations of two phase locking protocol

I. Deadlock

II. Cascading rollback ⟶ Phenomenon where a single transaction failure leads to a series of transaction rollback.

- Cascading rollbacks can be avoided by a modification of the original two phase locking protocol called strict two phase locking protocol. In this protocol, a transaction must hold all its exclusive ~~locks~~ locks till it commits or aborts.

- Rigourous two phase locking protocol is even stricter, which requires that all the locks be held until the transaction commits or aborts. In this protocol, transactions can be serialized in the order in which they commit.

Most database systems implement either strict or rigourous two phase locking

■ Lock Conversions

(i) lock Upgrade : This is the process in which a shared lock is upgraded to an exclusive lock

(ii) lock Downgrade : This is the process in which an exclusive lock is downgraded to a shared lock.

Page No:
Date:

Thus, the two phase locking protocol with lock conversions:

- First Conversion Phase:
  - Can acquire a lock S on item.
  - Can acquire a lock X on item.
  - Can convert a lock S to lock X (upgrade)

- Second Phase:
  - Can release a lock S
  - Can release a lock X
  - Can release a lock X to lock S (downgrade)

Timestamp Based Protocol

- Timestamp doesn't need any locks
  - Implies NO wait of transactions.

- Free from deadlock situation
  - Transactions are rolled backed and restarted in conflicting situations.

A timestamp is a unique identifier indicating the relative starting time of a transaction and is created by the database system.

- Approaches used for generating timestamps
  - (i) System Clocks
  - (ii) Logical Counter — # Incrementing every time a new transaction starts

The fundamental goal of timestamp protocol is to order the transactions globally in such a way that older transaction get priority in the event of a conflict.

Page No:
Date:

- TS(Ti) : Integer unique value
  Timestamp of transaction Ti before it starts its execution and assigned by database system.
  - The timestamps of the transactions determine the serializability order
  - If TS(Ti) < TS(Tj) then the produced schedule is equivalent to a serial schedule in which Ti appears before Tj
    i.e. < Ti , Tj >

- Properties of time stamp
  - Uniqueness - Each timestamp is unique
  - Monotonicity - Timestamp value always increase

→ Timestamp Ordering Protocol states that if Ri(x) and Wj(x) are conflicting operations then Ri(x) is processed before Wi(x) iff TS(Ti) < TS(Tj)

Whenever a schedule doesn't follow a serializability order according to the Timestamp, a user generally rejects it and rollback the transaction.

Some operations on the other hand are harmless and can be allowed.

Ti	Tj
R(X)	
	W(X)

* READ AND WRITE OPERATIONS OF DATABASE WITHIN THE SAME TRANSACTION MUST HAVE SAME TIMESTAMP

W-Timestamp (Q): Largest timestamp of any transaction that executed write (Q) operation successfully.

R-Timestamp (Q): Largest timestamp of any transaction that executed read(Q) operation successfully.

The R timestamp (Q) and W timestamp (Q) are updated whenever a new READ(Q) and WRITE(Q) instruction is executed.

T_1	T_2	Let TS(T1) = 1 and TS(T2) = 2
READ (A)		R-timestamp (A) = TS (T2) = 2
WRITE (A)		W-timestamp (A) = TS (T2) = 2
	READ (A)	
	WRITE (A)	

→ Timestamp Ordering Protocol

It ensures that any conflicting READ and WRITE operations are executed in timestamp order.

Suppose that Ti issues read (Q)

- If TS(Ti) < W-timestamp (Q), then Ti needs to read a value of Q that was already overwritten. Here the read operation is rejected and Ti rolled back.

- If TS (Ti) >= W-timestamp (Q), the read operation is executed and R-timestamp (Q) ← TS(Ti)

* TS(Ti) = 1 TS(T2) = 2 TS(T3) = 3

T_1	T_2	T_3	
Read (A)			R-TS(A) = 1
Write (A)			W-TS(A) = 1
	Read (A)		R-TS(A) = 2
	Write (A)		W-TS(A) = 2
		Read (A)	R-TS(A) = 3
		Write (A)	W-TS (A) = 3

Page No:
Date:

* TS(T1) = 1 TS(T2) = 2

T_1	T_2	T2 issues Read(A)
WRITE(A)		W-TS(A) = 1
	READ(A)	R-TS(A) = 2

Now TS(T2) >= W-TS(A)

⇒ Read(A) is executed and R-TS(A) ← TS(T2)

TS(T1) = 1 TS(T2) = 2

T_1	T_2	T1 issues Read(A)
	WRITE(A)	W-TS(A) = 2
READ(A)		

Now, TS(T1) < W-TS(A)

⇒ Read(A) is Rejected and T1 to be rolled back

Page No:
Date:

DATABASE RECOVERY

Process of restoring a database to the correct state in the event of a failure.

The recovery manager of the database system is responsible for ensuring two imp. properties of transactions:
- atomicity
- durability

- It ensures the atomicity by undoing the actions of transactions that do not commit.
- It ensures the durability by making sure that all actions of committed transactions survive in case of system crashes and media failures.

Shadow Copy Scheme

This scheme is based on making copies of database called 'shadow copies'

- It assumes that only one transaction is active at a time and the database is simply a file on disk.
- A pointer called db-pointer is maintained on disk; it points to the current copy of the database.

⊗ Unfortunately, the implementation is extremely inefficient in the context of large database since
(i) executing a single transaction requires copying the entire database
(ii) also the implementation doesn't allow transactions to execute concurrently with one another.

Page No:
Date:

Recovery facilities

(i) Backup mechanism: It makes periodic backup copies of the database

(ii) Logging facility: It keeps track of the current state of transactions and database modifications

(iii) Checkpoint facility: It enables updates to the database that are in progress to be made permanent.

(iv) Recovery management: It allows the system to restore the database to a consistent state following a failure

Log

Log is a sequence of log records, recording all the update activities in the database. It is kept on stable storage.

log records	Action
< Ti, start >	Transaction Ti has started
< Ti, O, OV, NV >	Transaction Ti has performed a write operation on the data item O. This data item O has old value OV before the write and a new value NV after the write
< Ti, commit >	Transaction Ti has committed.
< Ti, abort >	Transaction Ti has aborted.

Page No:
Date:

Database Updates / Modifications

Deffered Updates / Modifications

This scheme records all modifications to the log, but defers all the writes to after partial commit.

Actions taken during recovery after a crash

- Redone - iff both <Ti, start> and <Ti, commit> are there in the log, Execute redo (Ti) by updating db with new values

Immediate update / modifications

Allows database updates of an uncommitted transaction to be made as the writes are issued.

Update log record must be written before db item is written.

Recovery procedure has two operations instead of one:

(i) Undo (Ti) restores the value of all data items updated by Ti to their old value

(ii) Redo (Ti) sets the value of all data items updated by Ti to the new values.

Defferred database modifications

Ti	Primary Memory	Log	Database
Start executing		<Ti, start>	
Read (A)	A = 1000		
A = A - 50	A = 950	<Ti, A, 1000, 950>	
Write (A)			A = 950
Read (B)	B = 2000		B = 2050
B = B + 50	B = 2050		
Write (B)		<Ti, B, 2000, 2050>	
Commit		<Ti, commit>	

	Log	Action
	<T0, start>	
① →		No Action
	<T0, A, 1000, 950>	
② →		No Action
	<T0, B, 2000, 2050>	
③ →	<T0, commit>	Redo (T0)
	<Ti, start>	
	•	Redo (T0) & No Action (Ti)
④ →	<Ti, C, 650, 600>	
	<Ti, commit>	Redo (T0) & Redo (Ti)
⑤ →		

Case 1	Case 2	Case 3
<T0, start>	<T0, start>	<T0, start>
<T0, A, 1000, 950>	<T0, A, 1000, 950>	<T0, A, 1000, 950>
<T0, B, 2000, 2050>	<T0, B, 2000, 2050>	<T0, B, 2000, 2050>
<T0, B, 2000, 2050>	<T0, commit>	<T0, commit>
Failure	<Ti, start>	<Ti, start>
	<Ti, C, 650, 600>	<Ti, C, 650, 600>
	failure	<Ti, commit>
		failure
No action	Redo (T0) and No action for Ti	Redo (T0) and Redo (Ti)

IMMEDIATE DATABASE MODIFICATIONS

Ti	Primary Memory	Log	Database
Start executing		<Ti START>	
Read(A)			
A = A − 50	A = 1000	<Ti, A, 1000, 950>	A = 950
Write(A)	A = 950		B = 2050
Read(B)		<Ti, B, 2000, 2050>	
B = B + 50	B = 2000	<Ti, COMMIT>	
Write(B)	B = 2050		
Commit			

Failure point	log	Action
	<To, start>	
① →		No action
	<To, A, 1000, 950>	
② →		Undo(To)
	<To, B, 2000, 2050>	
③ →	<To, commit>	Redo(To)
	<Ti, start>	
④ →	<Ti, C, 650, 600>	Undo (To) & Undo (Ti)
	<Ti, commit>	
⑤ →		Redo (To) & Redo (Ti)

Case 1	Case 2	Case 3
<To, start>	<To, start>	<To, start>
<To, A, 1000, 950>	<To, A, 1000, 950>	<To, A, 1000, 950>
<To, B, 2000, 2050>	<To, B, 2000, 2050>	<To, B, 2000, 2050>
failure	<To, commit>	<To, commit>
	<Ti, start>	<Ti, start>
	<Ti, C, 650, 600>	<Ti, C, 650, 600>
	<failure>	<Ti, commit>
		failure
Undo(To)	Redo(To) & Undo(Ti)	Redo(To) & Redo(Ti)

Page No:
Date:

Multilevel Indexing

A multilevel indexing can contain any no. of levels, each of which acts as a non dense index to the level below. The top level contains single entry.

A multilevel index can be created for any type of first level index as long as first level index consists of more than one disk block

Advantage - It reduces the number of blocks accessed when searching a record, given its indexing field value.

* Because of the insertion and deletion problem, most multi level indexes use B-Tree or B^+ Tree data structures which leave space in each tree node to allow for new index entries.

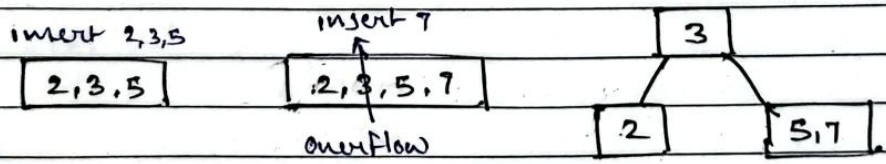
These data structures are variations of search trees that allow efficient insertion and deletion of new search values

In B Tree and B^+ tree data structures, each node corresponds to a disk block.

Each node is kept b/w half full & completely full.

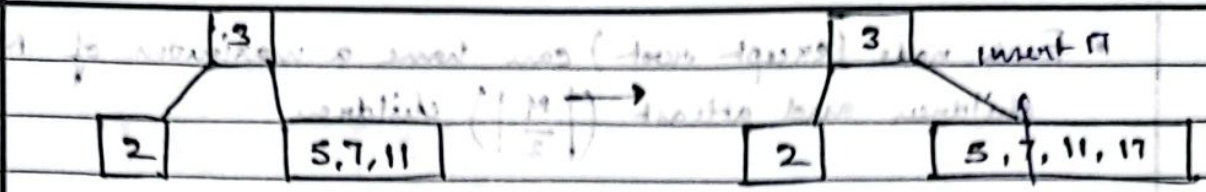
Q Construct a B tree of order 4

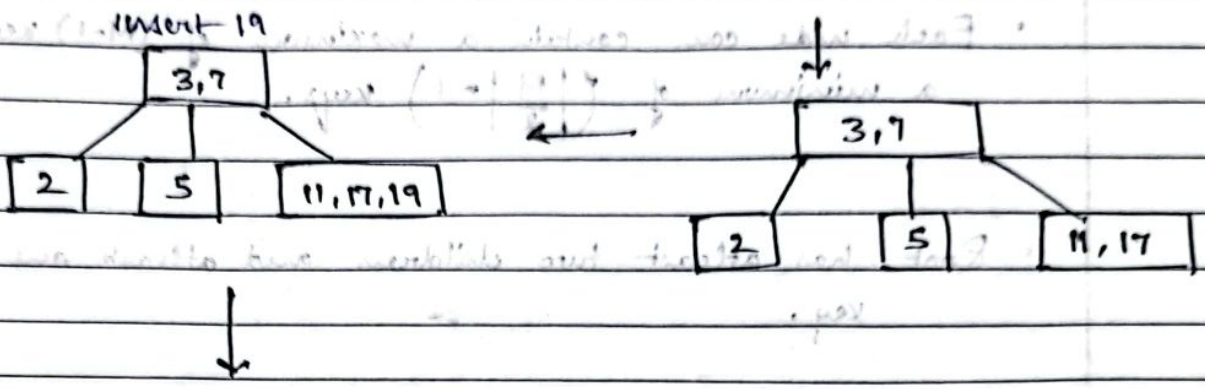
2, 3, 5, 7, 11, 17, 19, 23, 29, 31

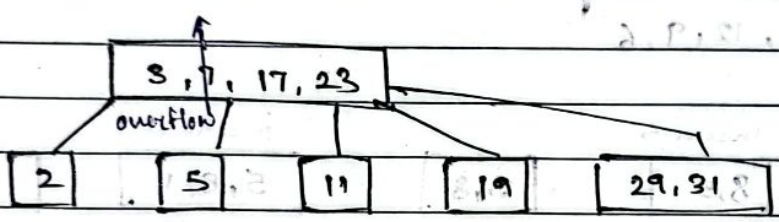


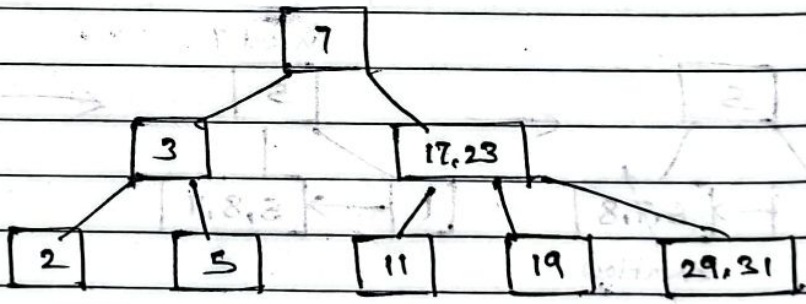
Page No:
Date:

Insert 11

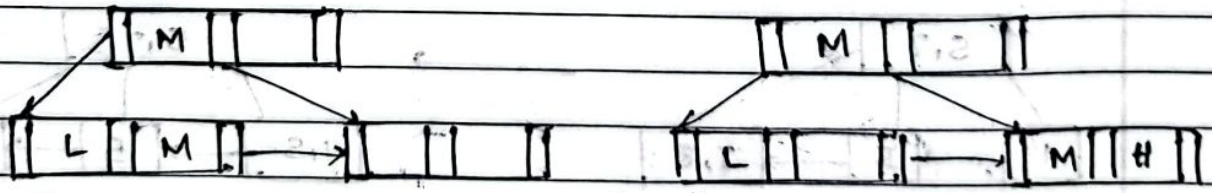








Building B+ tree



For a leaf node
If the node is full then classify the keys as L (lower), M (middle) and H (highest); then split the node.

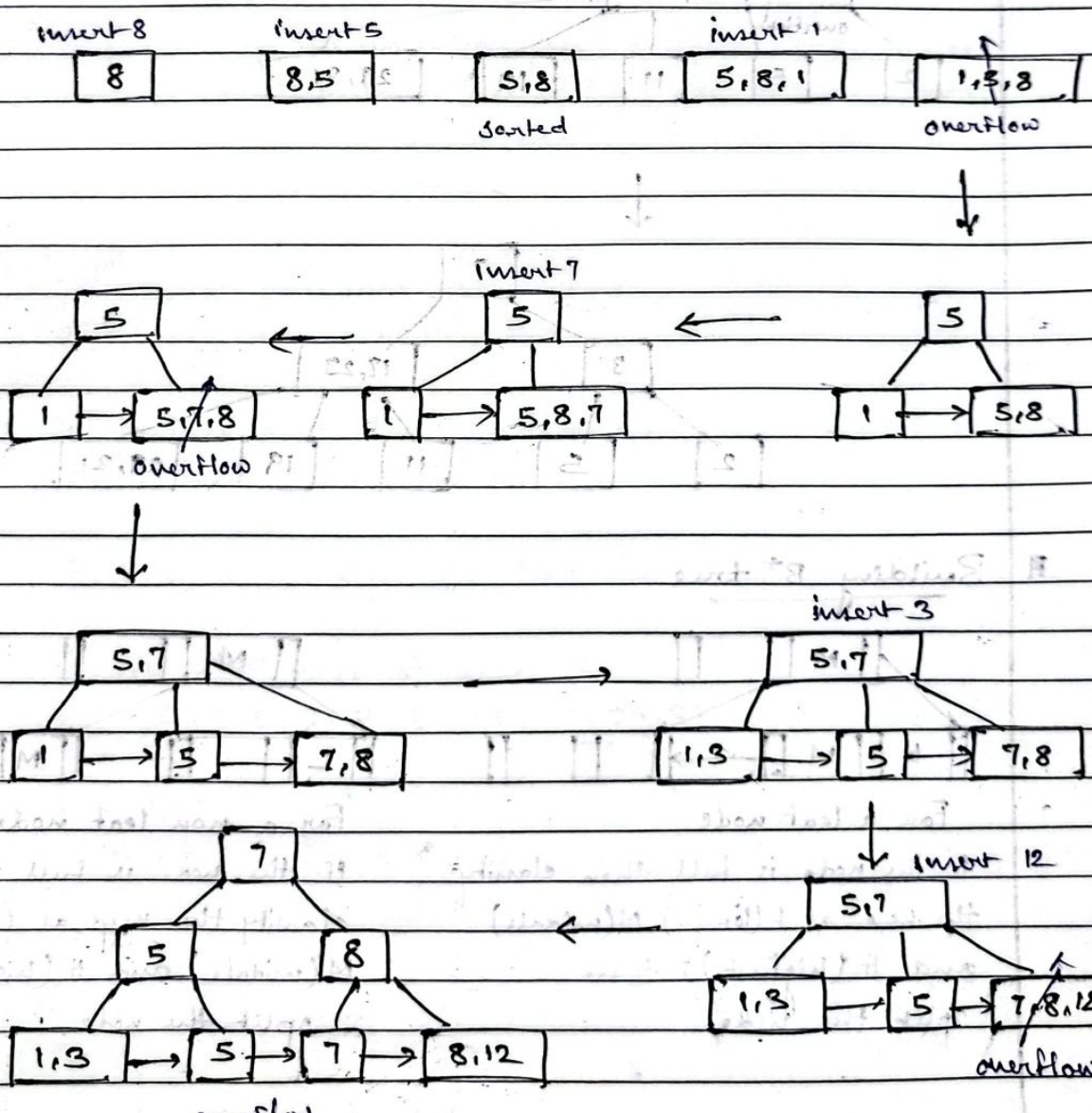
For a non leaf node
If the node is full then classify the keys as L (lower), M (middle) and H (highest); then split the node.

- Each node (except root) can have a maximum of M children and atleast $\left(\left\lceil \frac{M}{2} \right\rceil\right)$ children.

- Each node can contain a maximum of (M-1) keys and a minimum of $\left(\left\lceil \frac{M}{2} \right\rceil - 1\right)$ keys.

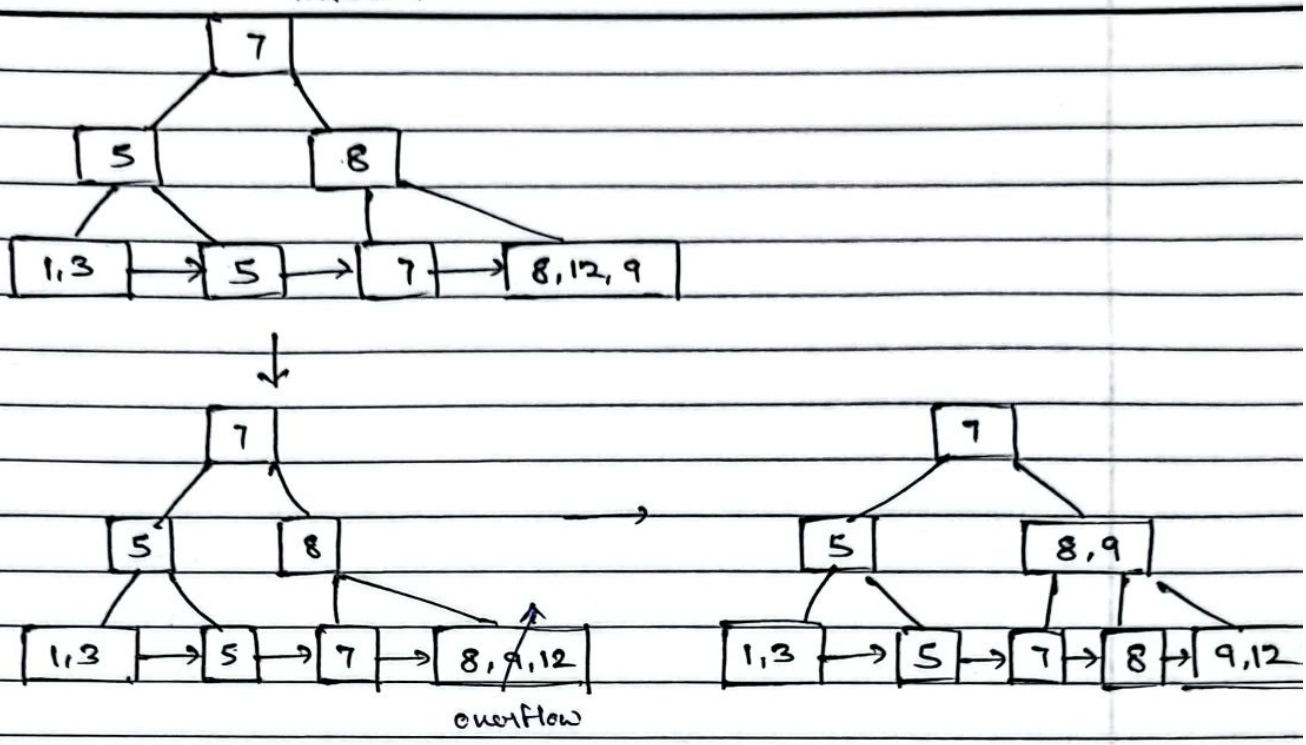
- Root has atleast two children and atleast one search key.

Q Construct a B^+ tree of order 3
8, 5, 1, 7, 3, 12, 9, 6



Page No:
Date:

Insert 9



Insert 6