CAP THEOREM:

- It is about how distributed database systems behave in the face of network instability.
- Any distributed database system can have atmost two of the following three properties:

* Consistency:

- It is about having a single, up-to-date, readable version of our data available to all clients.
- All data should be consistent.
- All writes are atomic and all subsequent request retrieve the new value.

* High availability:

- It states that the distributed database will always allow clients to select or update on items without delay.
- Request retrieves/receives a response whether it is success or failure.

* Partition tolerance:

- It is the ability of the system to keep responding to client requests even if there's a communication failure btw DB partition.
- System functions even if network between partition is temporarily lost.

Scenarios:

(i) In RDBMS, Reading and Writing happens on the same machine. So it is consistent but not partition tolerant because if the machine goes down there is backup.

(ii) System has two machines where both the machines read but only one machine can modify. The data takes time to reach from one system to another system so it inconsistent.

(iii) System has two machines, where one machine modify and another system backups the data. So it is consistent but not highly available.

DATA MODELING IN MONGODB

- Data has a flexible schema.
- Collections do not enforce document structure.
- The main challenge is balancing the need of application, the performance characteristic database engine & data retrieval patterns.

DATA MODEL DESIGN:

It has two types of data model:

* Embedded Data Model:

- Embed data in a single structure or document.
- They are generally known as "denormalized" models.

Eg:
```
{
    _id : <objectId1>;
    username : "123xyz",
    contact : {
                phone : "123-456-7890",
                email : "xyz@sample.com"
              },
}
```
Embedded sub-document

```
access : {
        level : 5,
        group : "dev"
    }
}
```

— Embedded sub-document

- It allows applications to store related pieces of information in same database record

USES:

- It is used when you have relationships between entities.
- It is used when you have one to many between entities.
- It provides better performance for read operations.

* Normalized Data Models:

- It describes relationships using references between documents

Eg:

```
{ _id : <objectId1>,
  username : "123xyz"
}
```

(contact doc)

```
{ _id : <objectId1>,
  username : "123xyz",
  phone : "123-456-7890",
  email : "xyz@sample.com"
}
```

```
{ _id : <objectId1>,
  username : "123xyz",
  level : 5,
  "dev" : group
}
```

(access doc)

Uses:

- It is used when embedding would result in duplication of data.
- To represent more complex M to M relationships
- To model large hierarchial data sets.

MONGODB CRUD OPERATIONS:

* Create Operations:
  - It add new documents to a collection.
  - If the collection doesn't currently exist, insert operations will create the collection.

Methods:

* db.collection.insertOne()

Eg: db.collection.insertOne(
{ name: "sue",
age: 26,
status: "pending"
}
)

* db.collection.insertMany()

Eg: db.products.insertMany([
{item: "card", qty: 15},
{item: "envelope", qty: 20},
{item: "stamps", qty: 30}
])

* Read Operations:

- It retrieves documents from a collection;
i.e; query a collections for documents.

METHODS:

* db.collection.find()

Eg: db.users.find(
{age: {$gt: 18}},
{name: 1, address: 1}
).limit(5)

* Update Operations:

- It modifies existing documents in a collection

METHODS:

* db.collection.updateMany()

Eg: db.users.updateMany(
{ age: { $lt: 18 } },
{ $set: { status: "reject" } }
)

* db.collection.updateOne()

Eg: db.inventory.updateOne(
{ item: "paper" },
{
$set: { "size.uom": "cm", status: "p" },
$currentDate: { lastModified: true }
}
)

* db.collection.replaceOne(

Eg: db.inventory.replace(
{ item: "paper" },
{ item: "paper", instock: [{ warehouse: "A",
qty: 60 }, { warehouse: "B", qty: 40 }] }
)

* Delete Operations:

. It removes documents from a collection.

METHODS:

* db.collection.deleteMany()

Eg: db.inventory.deleteMany({})

* db.collection.deleteOne()

Eg: db.inventory.deleteOne({status: "D"})

HBASE DATA MODEL:

. It is designed to handle semi-structured data that may differ in field size, which is in a form of data and column.

. It consists of various logical components:

* Table: It is made up of several columns
* Row: It consists of a row key and one or more associated value columns.
* Column: It consists of family of columns and a qualifier of columns (:)

* Column family: It physically position a group of columns and their values to increase its performance.

* Column Qualifier: It is added to a column family.

* Timestamp: It is written and is the identifier for a given version of a number.

HBASE CRUD OPERATIONS:

* General Commands:

- status: Display information cluster
  Eg - hbase(main):>status
- version: Shows version of hbase
  Eg - hbase(main):> version
- whoami: List the current user
  Eg - hbase(main):> whoami
- table_help: Gives reference shell command
  Eg - hbase(main):009:>table_help

* Create :

create 'employee', 'Personal info', 'Professional Info'

⇒ Hbase :: Table - employee

• Create a table with Namespace :

create 'company - empinfo : employee', 'Personal info', 'Professional Info'

• Put : Used to insert commands

put 'employee', 1, 'Personal info : empId', 10
put 'employee', 1, 'Personal info : Name', 'Alex'
put 'employee', 1, 'Professional Info : Dept', 'IT'

* Read : 'get' & 'scan' is used to read data

get 'table Name', 'Row key'
hbase(main) : 022 : get 'employee', 1

• Scan : Used to retrieve multiple rows

scan 'Table Name'
hbase (main) : 074 :> scan 'employee'

* Update : 'put' is used to update

put 'employee', 1, 'Personal info : empId, 30

* Delete : 'delete' is used to delete single cell.

delete 'employee', 1, 'Personal info : Name'

- Disable and drop :

disable 'employee'

drop 'employee'

Database Design:

The process of designing a database carries various conceptual approaches such as:

- Save disk by eliminating redundant data
- Maintains data integrity
- Provides data access
- Comparing Logical and Physical data models.

Logical data model:

It describes the data, without any concerns about physical implementations in the database.

Features:

- Each entity has well-specified attributes
- All the entities and relationships amongst them
- Primary key is specified for each entity
- Foreign key is used to identify relationship between different entities.
- Normalization occurs at this level

A logical model can be used the following approach:

- Specify all the entities with primary keys.
- Specify concurrent relationships btw entities
- Figure out each entity's attributes
- Resolve many-to-many relationship
- Carry out normalization.

Physical data model:

= It represents how the approach or concept of designing the database and shows the structure of the table.

Features:

=

- Specifies all the columns & tables
- Foreign keys defines relationship btw tables.
- De-normalization might occur.
- Straightforward reasons.
- It is different from RDBMS.

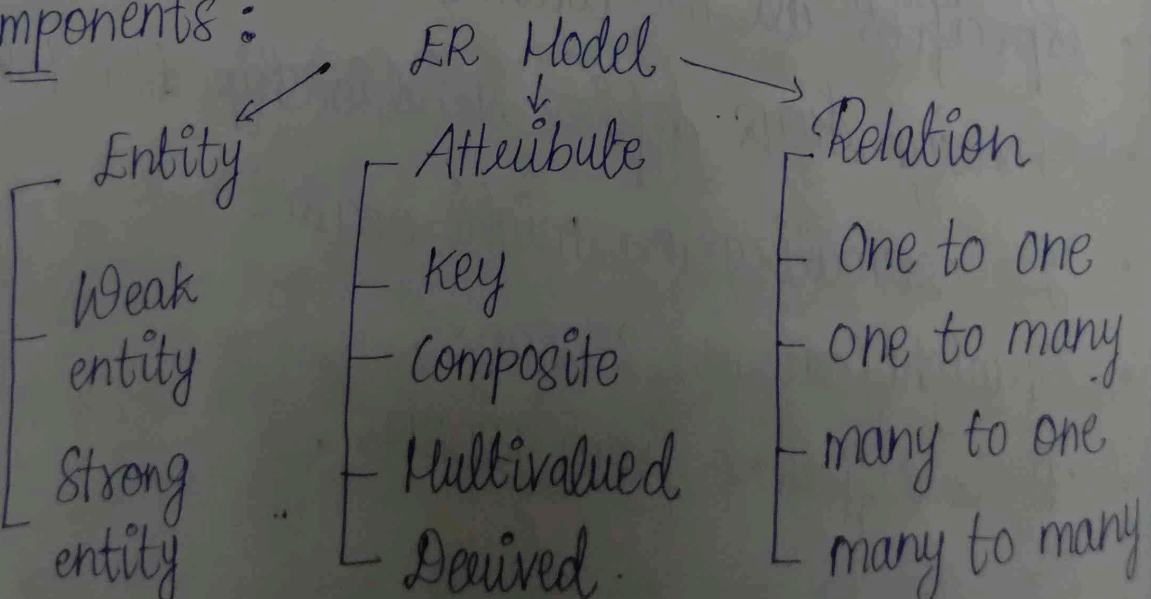
A physical model can be made using following approach:

- Convert the entities into tables.
- Convert the defined relationships into foreign keys.
- Convert the data attributes into columns
- Modify the data model constraints based on physical requirements.

ENTITY-RELATIONSHIP MODEL (ER MODEL):

- It is a very high-level data model.
- It is used define the data elements and relationship for a specified system
- It develops a conceptual design for database

Components:



* Entity:
= It may be an object, class, person or place. It is represented as rectangles. It has two types:

. Strong entity:
= It consists of key attributes forming a primary key. It is represented by rectangle.

. Weak entity:
= An entity which depends on another entity is called weak entity. It is represented by double rectangle.

* Attribute:
= It is used to describe the property of an entity. It is represented as eclipse.

Eg: name, age

Types:-

. key attribute:
= It is used to represent the main

characteristics of an entity. It represents a primary key.

- Composite attribute:

  It is composed of many other attributes.

  Eg: Name - First, Middle, Last Names

- Multivalued attribute:

  It can have more than values.

Eg: students having more than one Ph.No

- Derived attribute:

  It can be derived from other attributes

Eg: Age → DOB

* Relationship:

  It is used to describe the relation between entities. It is represented in diamond.

Types:-

- One to one:

  - Here only one instance of an entity is associated with the relationship

Eg: 1 female marry 1 male. vice versa

- One to Many:

  When one instance on left and more than one instance of right of an entity has relationship then it is called one to many.

Eg: Scientist can have many inventions

- Many to one:

  It is the vice versa of one to many relationship.

Eg: One course can have many students.

- Many to Many:

  When more than one instance on both left and right associates with relationship with then it is called as many to many.

Eg: Employee have projects, project can have many employees.

Extended ER Model : (EER Model)

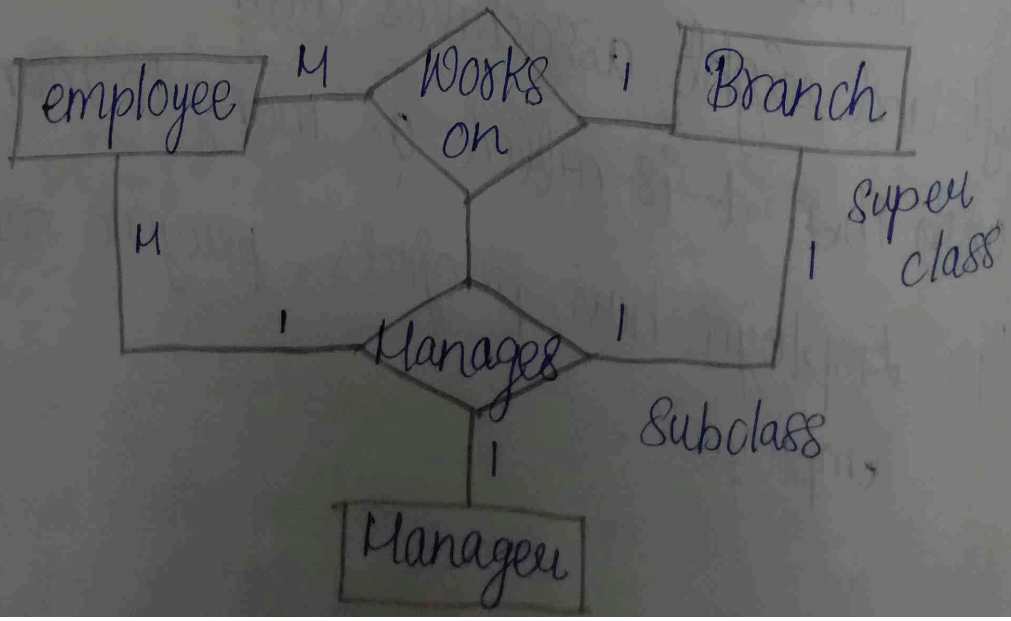
- It is a high level data model.
- It represents the requirements and complexities of complex database.

Types :

* Aggregation :

- It is an abstraction in which relationship sets are treated as high-level entity sets.
- It allows us to indicate that a relationship set participates in another set.
- It is used to simplify the details of a given database

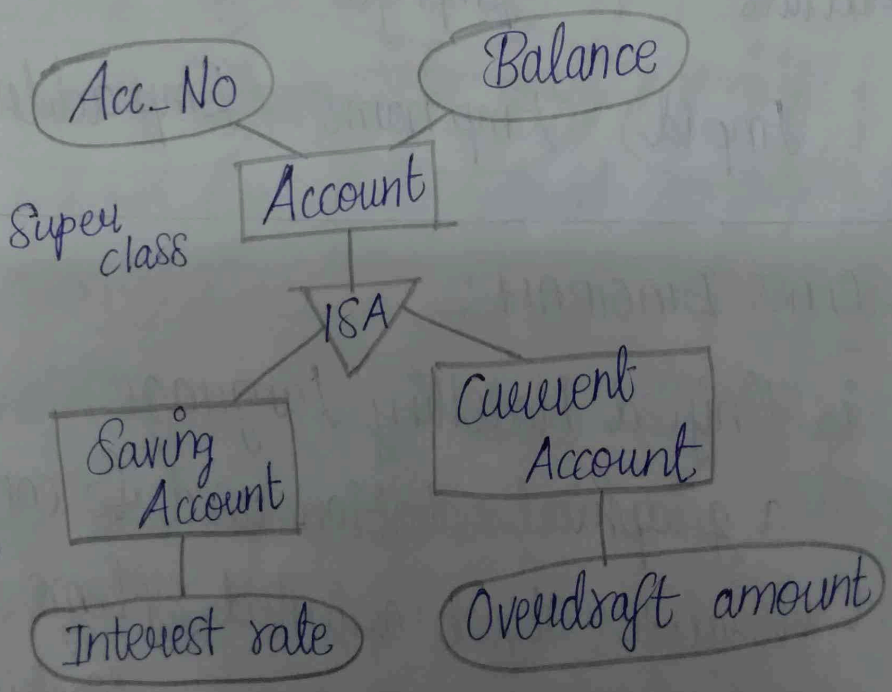
Eg :



* Specialization :

- It is the process of designing subgroupings with an entity set.
- It is a top-down process
- Subclasses or sub-entities can be formed from given attribute.
- It is represented by trangle labled ISA where ISA is refered as superclass - subclass relationship.

Eg:

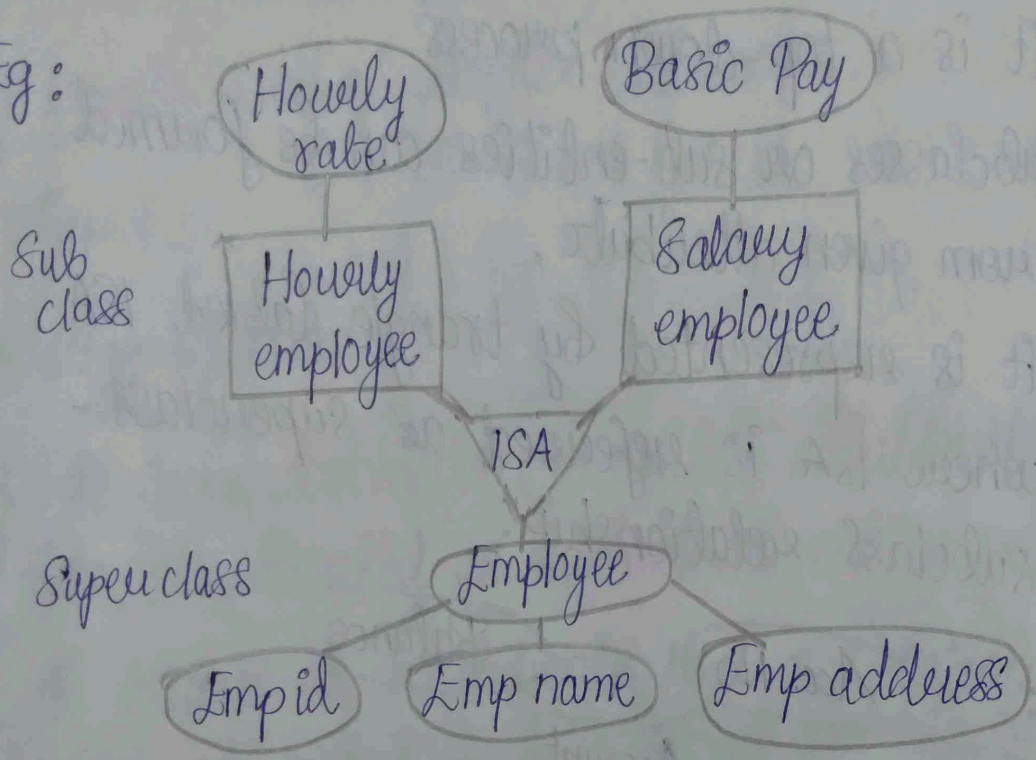


* Generalization :

- It is the reverse process of specialization.
- It is a bottom-up approach.

- It converts subclasses to superclasses.
- It combines a number of entity sets that shares the same features.

Eg:



UML CLASS DIAGRAM:

- It is Unified Modelling Language.
- It is a graphical notation used to construct and visualise object-oriented systems.
- It depicts a static view of an application

Class:

It is a blueprint of an object. Object and classes go hand in hand.

Class Name:

- The name of the class appears in the first partition

Class Attributes:

- Attributes are shown in second partition.
- Attribute type is shown after the colon.
- Attribute map onto member variables

Class Operations:

- Operations are shown in third partition.
- They are the services the class provides.
- The return type of a method is shown after the colon.

Class Visibility:

- +, - & # symbols before an attribute & operation name in a class denotes the visibility of attribute and operation.

Parameter Directionality:

Parameter is denoted as in, out, inout which specifies its direction to the caller.

This directionality is shown before the parameter name.

Relationships:

* Dependency:
  - It is a semantic relationship between two or more classes.
  - It forms a weaker relationship

Eg: Student (Name is dependent on Id)

* Generalization:
  - It is a relationship between a parent class and a child class.
  - Child class is inherited from parent class

Eg: Bank account - Savings, current & credit

* Association:
  - It describes a static or physical connection between two or more objects.
  - It depicts how many objects are there in the relationship

Eg: A department associated with college

Multiplicity :

- It defines a specific range of allowable instances of attributes.
- If range is not specified, it is considered as a default multiplicity.

Eg: Multiple patients are admitted to a hospital.

Aggregation :

- It is a subset of association
- It defines a part-whole or part of relationship

Eg: A company can exist without one employee

Composition :

- It is a subset of aggregation.
- It portrays the dependency between the parent and its child.

Eg: If you delete a contact book, all the contacts will be lost.

SQL DML:

- It is the language that gives users the ability to access or manipulate the condition the data model has inherited.
- Some SQL commands under DML
  * INSERT
  * UPDATE
  * DELETE

* INSERT: It is used to insert new rows or records in table.

Syntax: INSERT INTO TABLE_NAME (Column 1,... column N) VALUES (value 1, .., value N)

Eg: INSERT INTO EMPLOYEES (Emp_Id, Emp_Name) VALUES (04, "Harris")

* UPDATE: It is used to update or modify the value of a column in the table

Syntax: UPDATE table_name SET column N = value N WHERE [condition];

Eg: UPDATE Employees SET Salary = 1000 WHERE Emp_Id = 04;

* DELETE:

Syntax: DE
CO

Eg: DELETE

SQL DDL:

- It helps yo
- They are c
  modifying
- Some SQL co
  * CREATE *

* CREATE: It
in

Syntax: CREA
data

Eg: CREATE TA
Emp_Nam

* ALTER: It
of

* DELETE: It is used to remove one or more rows from the table.

Syntax: DELETE FROM Table_name WHERE condition;

Eg: DELETE FROM Employees WHERE Emp_Id = 04;

SQL DDL:

- It helps you to define the database structure
- They are capable of creating, deleting, and modifying data.
- Some SQL commands under DDL

  * CREATE * ALTER * DROP * TRUNCATE

* CREATE: It is used to create a new table in the database.

Syntax: CREATE TABLE table_name (column1 datatype, .. columnN datatype);

Eg: CREATE TABLE Employees { Emp_Id int(3), Emp_Name varchar(20) };

* ALTER: It is used to alter the structure of the database.

Syntax : ALTER TABLE table_name ADD columnName datatype ;

ALTER TABLE table_name DROP COLUMN columnName ;

Eg : ALTER TABLE Employees ADD BloodGroup varchar(255);

ALTER TABLE Employees DROP BloodGroup varchar(255);

* DROP : It is used to delete both the structure and record in the table.

Syntax : DROP TABLE Table_name ;

Eg : DROP TABLE Employees ;

* TRUNCATE : It is used to delete all rows from the table and free the space.

Syntax : TRUNCATE TABLE table_Name ;

Eg : TRUNCATE TABLE Employees ;

INTEG

. The

. It

info

. It e

othe

not

. It i

dama

TYPES :

* Doma

. It ca

valid

. It's

. The va

in the

Eg : ID

01

02

INTEGIRITY CONSTRAINTS:

- They are a set of rules.
- It is used to maintain the quality of information.
- It ensures data insertion, updating and other processes have to be performed by not affecting data integrity.
- It is used to guard against accidental damage to the database.

TYPES:

* Domain constraints:

- It can be defined as the definition of a valid set of values for an attribute.
- It's datatype includes string, character, etc.,
- The value of the attribute must be available in the corresponding domain.

Eg:

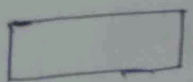
ID	NAME	AGE
01	Harris	18
02	Suresh	F → is an integer attribute

Not allowed as AGE

* Entity Integrity Constraints :

- It states that primary key value can't be null
- If primary key value is null, then we can't identify those rows.
- A table can contain a null value other than primary key field.

Eg :

Emp_Id	Emp_Name	Age
01	Harris	25
02	Suresh	26
	Sanjay	24

↓

Not allowed as primary key is null value / cant be

* Referential Integrity constraints :

- It is specified between two tables
- If foreign key in Table 1 refers to primary key of Table 2, then every value of Table 1 must be null or be in Table 2

Eg: Emp_name Name Age D_No → Foreign key

Table 1

101	Suresh	20	11
102	Haveis	21	24
103	Joshee	22	18 — Not allowed as 18 is not defined in Table 2
104	Sanjay	23	33

↕ Relationship

Table 2

D_No	D_Location
11	Tambaram
24	AHa Company
33	Mogappair

Primary key

* Key constraints:

- It is used to identify an entity within its entity set uniquely.
- A primary key can contain a unique value.

Eg:

ID	Name	Age
101	Suresh	18
102	Haveis	19
103	Joshee	20
102	Sanjay	21

↓

Not allowed as all rows must be unique.

INFERENCE RULE (IR):

- It is type of assertion. (FD)
- It can apply to a set of functional dependency to derive other FD
- We can derive additional FD from initial set.

TYPES:

* Reflexive Rule (IR 1):

If y is a subset of x, then x determines y

If $x \supseteq y$, then $x \rightarrow y$

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

* Augmentation Rule (IR 2):

- It is also called as Partial dependency.
- If $x \rightarrow y$, then $xz \rightarrow yz$

Eg: For R(ABCD), if $A \rightarrow B$ then $AC \rightarrow BC$

* Transitive Rule (IR 3):

- If $x \rightarrow y$ and $y \rightarrow z$ then $x \rightarrow z$

* Union

- If $x \rightarrow$

Proof:

1) $x \rightarrow y$
2) $x \rightarrow z$
3) $x \rightarrow x$
4) $xy \rightarrow y$
5) $x \rightarrow y$

* Decompos

- It is al is the
- If $x \rightarrow$

Proof:

1) $x \rightarrow$
2) $yz \rightarrow$
3) $x \rightarrow$

* Pseudo

* Union Rule (IR4):

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

Proof:

1) $X \to Y$	Gn
2) $X \to Z$	Gn
3) $X \to XY$	Using IR2 on 1
4) $XY \to YZ$	Using IR2 on 2
5) $X \to YZ$	Using IR3 on 3 & 4

* Decomposition Rule (IR5):

. It is also known as project rule and it is the reverse of union rule.

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

Proof:

1) $X \to YZ$	Gn
2) $YZ \to X$	using IR1
3) $X \to Y$	using IR3 on 1 & 2

* Pseudo Transitive Rule (IR6):

- If $X \to Y$ and $WY \to Z$, then $XZ \to W$

Proof:

1) $X \to Y$ — gn
2) $WY \to Z$ — gn
3) $WX \to WY$ (Using IR2 on 1 with W)
4) $WX \to Z$ (using IR3 on 3 & 2)

MINIMAL COVER:

- It is a set of FD.
- E is a minimal set of FD, F is equivalent to E
- A set of FD F to be minimal if it;
  - ⇒ Every dependency in F has a single attribute for its right-hand side
- It is also called canonical cover
- A set of FD FC is called canonical cover of F if each FD in FC is a;
  - ⇒ Simple FD
  - ⇒ Left reduced FD
  - ⇒ Non-redundant FD

Eg: Consider an example to find canonical covers of F.

The given FD are as follows:

$A \rightarrow BC$

$B \rightarrow C$

$A \rightarrow B$

$AB \rightarrow C$

Minimal Cover: It is a set of FDs which are equivalent to the given FDs.

canonical cover: The LHS must be unique

Step 1: Convert RHS to singleton attribute

$A \rightarrow B$

$A \rightarrow C$

$B \rightarrow C$

$A \rightarrow B$

$AB \rightarrow C$

Step 2: Remove extra LHS attribute

So $AB \rightarrow C$ can be converted into $A \rightarrow C$

$A \rightarrow B$

$A \rightarrow C$

$B \rightarrow C$

$A \rightarrow B$

$A \rightarrow C$

Step 3 : Remove the redundant FDs

$A \rightarrow B \quad A \rightarrow C$

$B \rightarrow C$

Now we will convert these into canonical cover.

$A \rightarrow BC$

$B \rightarrow C$

PROPERTIES OF RELATIONAL DECOMPOSITION :

* Attributes Preservation :

⇒ Using FDs the algorithms decomposes the universal relation schema R in a set of relational schemas $D = \{R_1, R_2, \ldots, R_n\}$, where D is called Decomposition of R.

⇒ The attributes in R will appear in the decomposition i.e., no attribute is lost. This is called the Attribute Preservation.

* Dependency Preservation :

⇒ If each FD $X \rightarrow Y$ specified in F appears directly in one of the R_i in the Decomposition

⇒ Or could be inferred from dependencies that appear in some R_i, this is Dependency Preservation.

Eg: $R = (A, B, C)$

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

Key = $\{A\}$

R is not in BCNF

Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$

* Non-Additive Join Property:

⇒ It ensures no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations resulting from decomposition.

* No redundancy:

⇒ Decomposition is used to eliminate problems of bad design like anolomies, inconsistencies and redundancy. If there is no proper decomposition, it may lead to loss of information.

* Lossless join:

⇒ It is a feature of decomposition supported by normalization.

⇒ It is the ability to ensure that any instance of original relation can be identified from smaller relations.

Eg: A decomposition is loseless if we can recover:

$R\{A,B,C) \rightarrow$ Decompose $\rightarrow R_1(A,B)\ R_2(B,C)$

$\rightarrow$ Recover $\rightarrow R'(A,B,C)$

Thus $R' = R$ (R = lossless decomposition)

NORMALIZATION:

- It is the process of organizing the data.
- It is used to minimize the redundancy from a relation or set of relations.
- It is also used to eliminate - Insertion, Update and Deletion anomalies.
- It divides the larger table into smaller table and links them using relationship.

- The n
redu

TYPES:

* INF

- A rel
an ato
- It stab
multip
- It can

Eg: ID

1

2

3

Table 2

- The normal form is used to reduce redundancy from the database table.

TYPES:

* 1NF : First Normal form
- A relation will be in 1NF if it contains an atomic values.
- It states that an attribute cannot hold multiple values.
- It can hold only single-valued attribute.

Eg: Table 1

ID	Name	Courses
1	A	C_1, C_2
2	E	C_3
3	M	C_2, C_3

Table 2

ID	Name	Course
1	A	C_1
1	A	C_2
2	E	C_3
3	M	C_2
3	M	C_3

* 2NF : Second Normal Form

- It must be in relational with 1NF
- In 2NF, all non-key attributes are fully functional dependent on the primary key.

Eg: ②

Table 1

STUD_NO	COURSE
1	C_1
2	C_2
1	C_3
4	C_4
4	C_1

Table 2

COURSE_NO	COURSE_FEE
C_1	1000
C_2	1500
C_3	1000
C_4	2000
C_5	2000

①

STUD_NO	COURSE_NO	COURSE_FEE
1	C_1	1000
2	C_2	1500
1	C_4	2000
4	C_3	1000
4	C_1	1000
2	C_5	2000

* 3NF : Third Normal Form

- It is relation with 2NF and doesn't contain transitive partial dependency.

. It is used to reduce data duplication.
. It is also used to achieve data integrity.

Eg: Consider relation R(A, B, C, D, E)

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

All attributes on right sides of all FDs are prime.

* Boyce-Codd Normal Form (BCNF)

. It is the advance version of 3NF. It is stricter than 3NF.
. The table should be in 3NF, and for every FD, LHS is super key.
. It is free from redundancy

Eg: Consider relation R(A, B, C)

$A \rightarrow BC$

$B \rightarrow$

A & B both are super key so this relation is BCNF.

TRANSACTION:

- Concurrency control deals with interleaved execution of more than one transaction.
- A set of logically related operations is known as transaction.
- The main operations are:

* Read(A): R(A) reads the value of A and stores it in a buffer in main memory
* Write(A): W(A) writes the value back to the database from the buffer.

Let us take a debit transaction:

1. R(A);
2. A = A-1000;
3. W(A);

Assume A=5000,

- The first operation reads the value of A from database and stores it in a buffer.
- The second operation will decrease its value by 1000. So buffer will contain 4000.

. The third operation will write the value from the buffer to database as 4000.

The transaction may fail after executing some of its operations. The failure can be because of hardware, software or power, etc. To avoid this, database has two operations:

* Commit: After all instruction of transaction are successfully executed the changes made by a transaction is made permanent in DB
* Rollback: If a transaction is not able to execute all operations successfully, all changes made by a transaction are undone.

ACID PROPERTIES IN DBMS:

. A transaction is a single logical unit of work that accesses and modifies the concepts of database.
. Transaction accesses data using read and write operations.

ACID
- Atomicity : Entire transaction takes place once or doesn't happen at all.
- Consistency : DB must be consistent before and after transaction.
- Isolation : Multiple Transactions occurs independently without interference.
- Durability : Changes of a successful trans. occurs even if the system failure occurs.

PROPERTY	RESPONSIBILITY
Atomicity	Transaction Manager
Consistency	Application Programmer
Isolation	Concurrency control manager
Durability	Recovery Manager

* Atomicity :
- As a transaction is a set of logically related operations. either all them should be executed or none.

: It has two operations :

→ Abort : If transaction aborts, changes made to the database are not visible.

→ Commit : The changes are visible in database

* Consistency :

. Integerity constraints should be maintained so that transaction is consistent before and after the transaction.

. It refers to the correctness of a database

* Isolation :

. It ensures that multiple transaction can occur concurrently without any interference.

. It ensures that the result state is achieved.

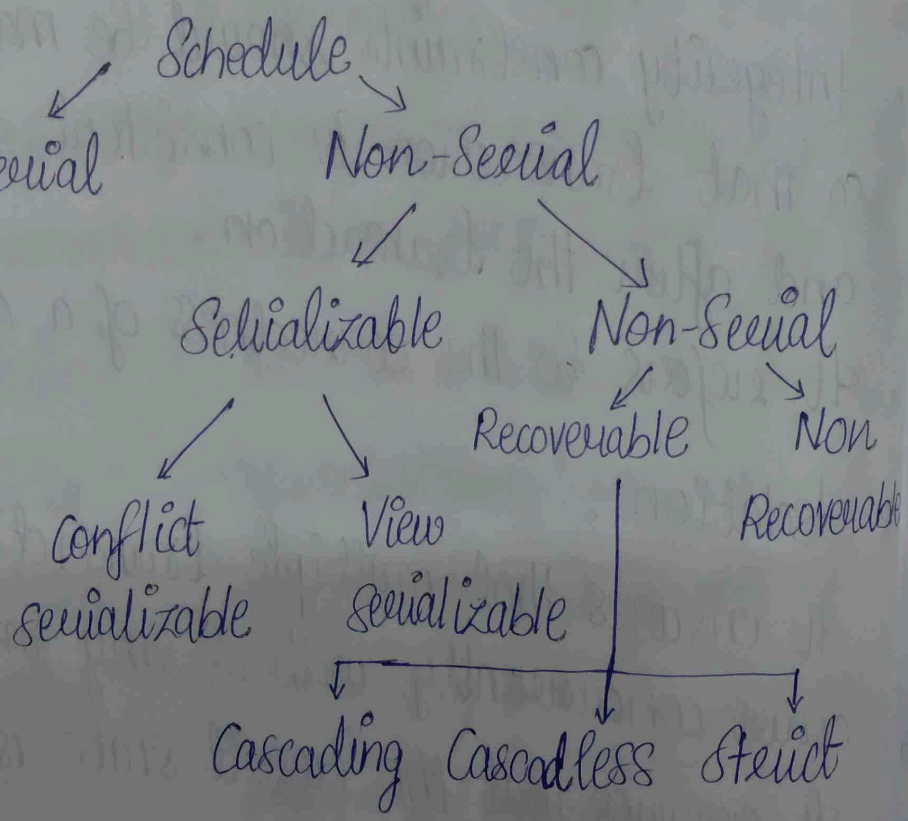
* Durability :

. Once the transaction is over, it ensures the updates and modifications to the database are stored and written to the disk.

. It is stored in non-volatile memory.

SCHEDULES :

- It is a process of lining the transactions and executing them one by one.
- It is brought into play and the transactions are timed accordingly.



TYPES :

* Serial Schedule :

- No transactions starts until a running transaction has ended.
- Transactions are executed non-interleaved.

Eg :

* Non-
- Opera
interle
- It lea
- The tr
for pe
- It is

a) Seria
- It is u
the dat
- It is us
to incon
- It has

Eg:

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

* Non-Serial schedule :

- Operations of multiple transaction are interleaved.
- It leads to concurrency problem.
- The transaction proceeds without waiting for previous transaction to complete.
- It is further divided into :

a) Serializable :

- It is used to maintain the consistency of the database.
- It is used to verify whether it will lead to inconsistency or not.
- It has two types :

(i) Conflict Serializable:

- It can be performed by swapping non-conflicting operations.
- It is conflict equivalent to a serial schedule.

(ii) View Serializable:

- It is view equal to a serial schedule.
- If the serializability contains blind writes, then view serializable doesn't conflict serializable.

b) Non-Serializable:

- It is divided into two types:

(i) Recoverable schedule:

- Transactions commit only after all transaction whose changes they read commit.
- It has three types:
  - ⇒ Cascading schedule
  - ⇒ Cascadless schedule
  - ⇒ Strict schedule

(ii) Non-recoverable schedule:

- A transaction does a dirty read operation from an uncommited transaction and commits before the transaction from where it has read value.

Examples:

Non-Serial schedule:

T_1	T_2
$R_1(A)$	
$W_1(A)$	
	$R_2(B)$
	$W_2(B)$
$R_1(B)$	
$W_1(B)$	
$R_1(B)$	

View Serializability:

Rules:

- If T_1 reads initial value of A then T_2 does the same
- If T_1 reads the written value then T_2 does the same...
- If T_1 writes the final value then T_2 does the same.

Conflict Serializability: (operations)

• Read i(x) read j(x)	Non-conflict	R-R operation
• Read i(x) write j(x)	Conflict	R-W operation
• Write i(x) read j(x)	conflict	W-R operation
• Write i(x) write j(x)	conflict	W-W operation