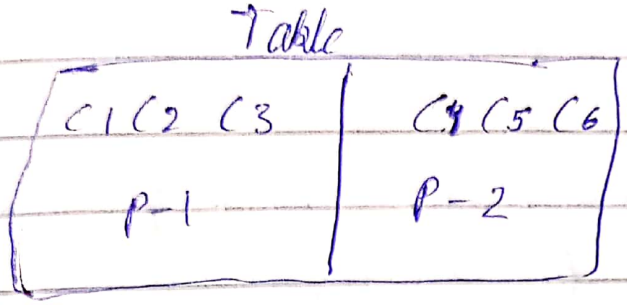
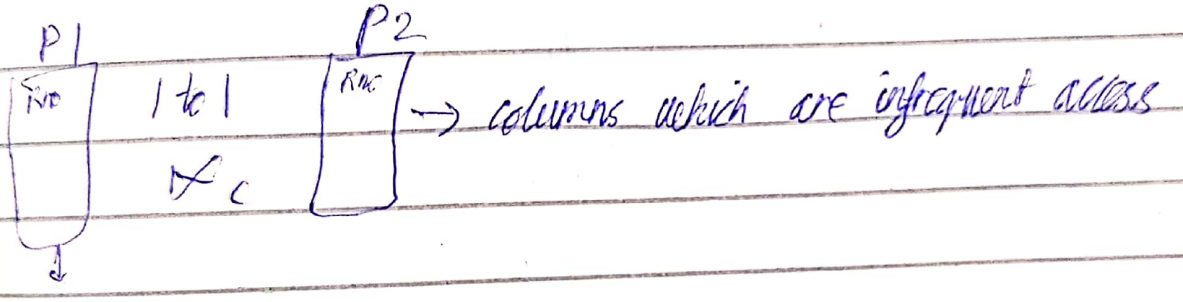
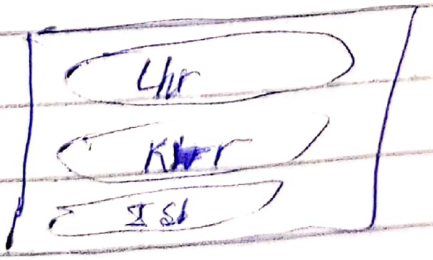
You can create vertical partitioning of table to improve performance





Columns which are frequently access is in the partition 1

In horizontal partitioning you have to specify range or expression. you can select partitioning criteria based on some attribute e.g on campus



Both techniques are not mutually exclusive. We can build the hybrid with both horizontal & vertical to increase the performance.

Normalization: to reduce data redundancy
Partitioning: to improve performance

Performance always improve if we either access partition 1 or partition 2.
When we want data from both partitions, so we have to join them. So overhead is joining

If we add in frequent attribute in P1 so it may improve queries but it can have bad impact on performance of most frequent access queries.
We have to see the performance of most frequently access queries.

You can revise your decision based on performance of queries.

Designing is a continuous process

index on partitions

DBMS: column based storage system
document " "
record " "

NoSQL systems are used to handle the big data
Big data is problem
NoSQL system is solution

Note that PK must be present in both partitions

Covered index :

not a new technique of indexing

a) if your query sends the result by just scanning index table means without accessing base table, the index behaves like a covered index

Storage wise partitioning is better

mostly queries may require some columns, not all

evenly distributed advantage :

- node balancing
- we get result in efficient time

In range partitioning, you can avoid unnecessary data access.
In Hash based partitioning, we get parallel access.

Hybrid :

Firstly do Hash partitioning, then do range partitioning, here we can avoid unnecessary data access and get parallel access

No comparison of Traditional views with Materialized views

Materialized Views (MV)

VS

Traditional Views (Virtual Table) // purpose is not a performance

Traditional views required data storage but temporary, when it is materialized at runtime.
We should have enough space for views at runtime.

PRE-JOIN }
PRE-Agg } usage:
if they are some data structure, they have some frequent of access and it requires lot of joins.

(MV), (View), (Tables)
Summary table : one time join and aggregation, it stores the data in table (~~of~~ when we do PRE Join & Agg)
There is a need of user defined programs (development cost of maintenance)
→ storage and maintenance cost

MV:
it requires storage cost and maintenance cost, and performance improves.
There is no need of user defined programs, when ~~the~~ there is an update in other tables. (no development cost of maintenance)
You can maximize performance and you can develop flexible structure with the help of normalized structure
consistency is automatically maintained by DBMS.

Q: which require data from all dimension & facttess table with same aggregation?

→ we can use summary table

If we want ~~from~~ data from facttess table with one dimension

→ You have to disclose physical data model.

→ less flexibility

→ ~~We have to train the user~~

→ inform the user when you drop summary table

With the help of MV:

→ transparent from end user

→ it rewrite the query (auto rewrite)

→ auto consistency

→ maximum flexibility, at any time you can drop the MV but there is no need to uniform user.

There is a comparison of MV with summary table

On top of table you can create MV

Data marts have high performance, because they have summary data already available.

Through MV, we get performance of denormalized structure and flexibility of normalized structure

MV: advanced version of pre-join & pre-aggregation technique

If you have mart data from summary table so you have to refer it. In MV, there is no need to refer it a query optimizer may decide ....

~~physical~~ physical structure by end user

Q1 : 3D (Most frequently) Q2 : 4D (low frequency)

a) There is a MV having 3D so what we have to do 4D.
If we create new MV by extending old MV so it would effect the performance of Q1.

b) If we create another MV without extending 3D MV, So this MV have storage and maintenance cost

c) We can join old MV (with 3D), with 4th Dimension, we have avoid storage cost, and avoiding 3 expensive join operation, we will compromise some performance, we have also avoided maintenance cost here.

You can also join :

a) MV with MV
b) MV with table

You can also create index of MV

MV have 4 flavours :

1. Join Indexing } those are common to
2. Aggregate Join Indexing } all type of DBMS architecture
3. Reverse Tables } issues
4. Replicated Tables } in shared nothing architecture

MV name is not standard

DBMS Architecture :

a) Shared everything architecture (CPU & memory is shared)
b) Shared nothing // (every CPU has its own memory)
it can communicate through protocol
all nodes perform similar task parallely

A MV can be based on different MV, but on leaf, it based on tables

shared everything arch: different nodes perform different tasks.

Reference of locality high → Performance high if Product Desc & Product Sales are on same node

Remotely access (TCP/IP)

↓

performance poor if Product Desc & Sale is on different nodes.

We have to bring Product Sale data on the node which has product desc we bring it in his cache.

Reversal Tables: (Table Redistribution)

Replicated Tables:

→ additional storage cost

→ We can replicate product table in all nodes and we can join it with specific Sales info stored in that respective node.

Reversal Tables:

→ additional storage cost

→ no aggregation no joining

Using these emerging function we can get data by just scanning large table only one time.

In grouping set, you can specify any combination.

Roll up (D1, D2, D3)

Cube # it gives all combinations

We use grouping set, when we want specific combinations only

Data Mining Basics use the mathematical algo to extract pattern, driven by KDD.

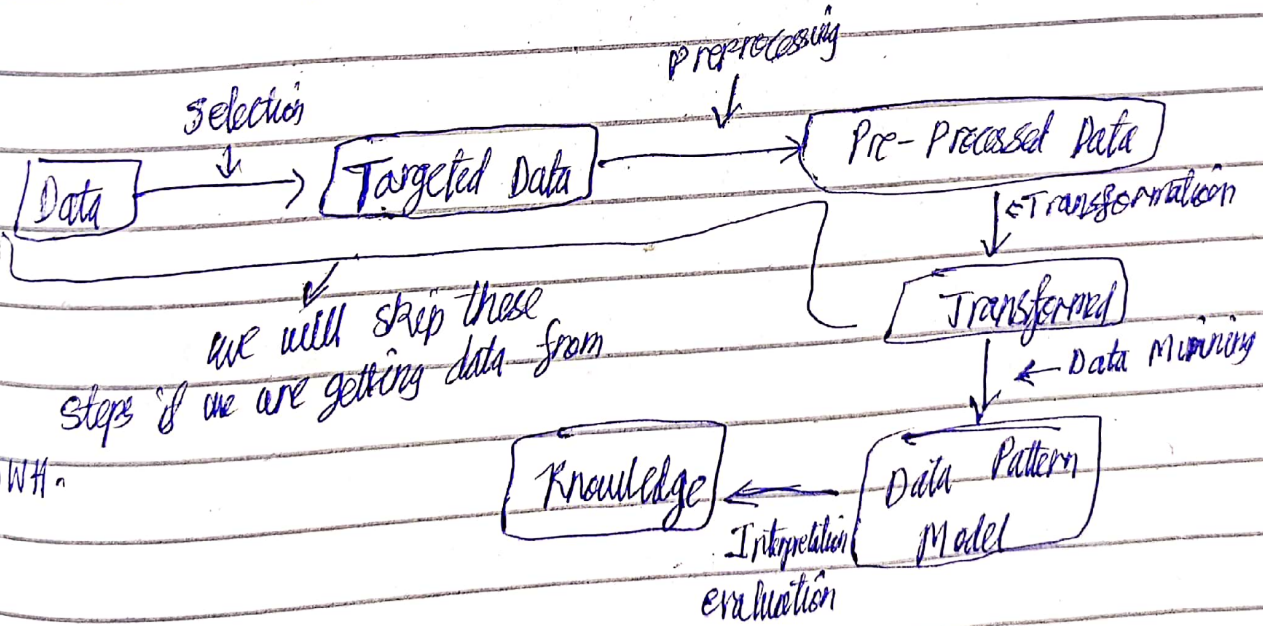
vs

Knowledge Discovery in Data (KDD) process to find hidden trends, patterns useful info model

a) valid
b) useful
c) unexpected
} Data Mining Queries Result

In OLTP, we get subset of data ~~from~~ from database

In data mining we have pre define data.



Data mining gives unseen and unexpected data

In classification, we map objects onto pre defined classes.

Data Mining use data mining algo to extract model, patterns within Knowledge discovery process.

Architectures :

- No coupling
- Loose coupling
- Semi-Tight coupling
- Tight Coupling

In no coupling, data is in files, no database, no data warehouse. It is not utilizing the functions of database or DWH.
File based system

In loose coupling, data is retrieved or stored in Database/DWH

Semi tight coupling :
Integrate data mining system with DB/DW to use some functionality of DB/DW, not all.

Tight coupling :
DW and Data Mining is the part of the architecture it is an integral component which will use all functionalities.
-) more performance
-) max scalability
Used for large data

Applications :

everywhere in the world (Health, Banking

Data Mining properties :

- valid
- understandable
- useful
- unexpected

Threshold is defined by you

Adv & Disadv

Personal

- Privacy (data is disclosed)
- Security (People can Hack it)
- Illegal use of information (unauthorized use)

- processes : CRISP DM (Cross Industry standard process Data Mining)

Phases : - Business understanding → Data understanding → preparation ~~Pre-processing~~ data → Data Mining → Evaluation → Deployment

We use it for (Sales & Promotion)

- Techniques :
  - Association → ~~finds relationship~~
  - classification → pre defined classes
  - clustering → first identify the class then map the objects onto the classes
  - prediction → find relationship b/w dependent & independent variable e.g Sales & Profit

Clustering : e.g market segmentation, document
all books which have similar characteristics are colocated.

interesting rules :
Rules which are defined fulfilling threshold (confidence level)

itemset : A collection of ~~may contain~~ 1 or more items.

K - itemset :
→ An itemset contains K items

Support count (σ)
→ Frequency of occurence of an itemset
e.g $\sigma(\{Milk, Bread\}) = 2$

Support :
→ $P(X)$?

$$\frac{\sigma(X)}{|T|}$$

Fraction of transactions that contain an itemset

Frequent Itemset :
→ An itemset whose support is greater than or equal to a min support threshold / min confidence

If there are 5 items & itemset = $2^5 - 1$ (ignore NULL set)

$$P(Y|X) = \frac{P(X \cup Y)}{P(X)}$$

Association Rule

- An implication $X \rightarrow Y$, where X and Y are itemsets
- e.g {Milk, Diaper} → {Cereal}

min_support

min_confidence

$A \rightarrow C$

support = support({A} ∪ {C}) = 50%

confidence = support({A} ∪ {C}) / support({A}) = 66.6%

if A B are frequent, then its all subset must be frequent

if an itemset is defined infrequent, so don't check its superset. it will be infrequent also.

Frequent Itemset :

Illustration of the Apriori Principle.

any subset of a frequent itemset must be frequent

~~Big Data~~

-) operational system, which contains business data
-) Data Warehouse, normally used for data analytics
-) NOSQL DB is Big Data Storage Sys → Raw data

We make requirement driven strategies.

Big Data

is a problem and NOSQL system is solution of Big Data

Data Integrity
Consistency
Concurrency
} OLTP requirement / provides

In NOSQL :

- availability is major thing (Data should be available)

Replication of Data :

Cost :

-) Storage
-) To maintain consistency
  ↓
  we need master slave architecture for it.

DW can store both structure & unstructured data