video#1

Database ⇒

1.Relational (sql)
2.Graph
3.Key-value
4.Column-store

==========

Relationa (sql) most popular.

==========

Table ⇒relation
Column ⇒attributes
Row ⇒ tuples ⇒record

==========

Schema ⇒ definition of table
Consist of :

⇒ columns name
⇒ datatype of columns
⇒ primary keys
⇒ nulls columns
Etc..

==========

video#2

Relational algebra ⇒
One of the theories that sql was based on.
Operation we need for tables (language).
Kinds of algebra ⇒
1.Elementary algebra (+ - *)
2.Boolean algebra (and ,or ,not)
3.Relational algebra

==========

Relation ⇒
Represent a kind of entity / object with certain attribute
e.g.
car(model , year , color)

Relational algebra:
1) **Selection** ⇒output tuples satisfy a condition σ
2) **Rename** ⇒output a relation with rename attributes ρ
3) **Projection** ⇒output a relation has only the specific att , can add ,remove and change att
order. π
4) **Union** ⇒output tuples that appear in one or both of the input relation .duplicates are
remove ,input must have same att ∪
5) **Intersection** ⇒ output tuples that appear in both of the input relation .input must have
same att ∩
6) Difference ⇒output tuples that appear in the first but not in the second input relation.,input must have same att -
7) **Product** ⇒output all possible combinations of tuples from the input relation ×
8) **Join** ⇒

⇒ Relational algebra تعتمد علي ال set
ومفهوم ال set
⇒No duplicate
⇒Order not necessary

⇒Relational algebra تحدد ترتيب الخطوات
⇒ input و output عبارة عن relation
⇒query بين ال combination نعمل
==========

video#3

Storage types

Cpu register
Cpu cache
Dram

features

Volatile , random access , bytes , smaller , faster , more expensive
==========

Ssd
Hdd
Network storage
Features
non-Volatile , sequentioal access , block/page , larger , slower , cheaper

Access times

Dram ⇒ 100ns
Ssd ⇒ 16000ns
Hdd ⇒2000000ns
Network ⇒ 50000000ns

Design goals

⇒ mange data that exceed the available memory
⇒ minimize reading/writing to disk
⇒ when access data on disk minimize sequential accessDatabase storage
Consist of database files
System specific file formate
Os doesn’t understand the content

What is pages?

⇒Fixed-size block of data.
⇒Can contain anything
tuples,meta-data,index,log record
⇒Unique id for each page.
⇒Dbms can map a page id to a physical location.

File organization

1.heap file organization.
2.tree file organization.
3. Sequential file organization.
4.hashing file organization.

Heap file.

⇒Page directory
1. Special page within each file
2. Location of each data page in the file
3. Number of free “slots” per page
4. List of free / empty pages
5. Must be kept in sync with data page

==========

Page header

1.exists in every page
2.metadata
.page size
.compression/encoding info
.dbms version

==========

video#4

Differents ways to store info in pages ⇒

1.slotted pages

Header maintains:
#slotes
Offset of last used slotes.
Each slotes pointes to corresponding tuple.Tuple are added starting from the end of the page.
E.g. postgre sql .. sql server

2.log-structured storage (log-structured merge tree lsm tree)

Recorde ”change” rather than storing the tuples in pages
Each entry : set or delete operation on a tuples must include the tuple id
E.g.cassandra .. rocks db .. hbase

3.index organized tables

Store the actual data inside an index
Faster reads
Slower writes
Reduced storage
e.g.my sql .. sql server .. oracle

==========

Record ID = tuple ID ⇒
Unique for every tuple .

Represents physical location of tuples.
includes(file id,page id ,slot#).
Usually not stored with the tuple.
==========

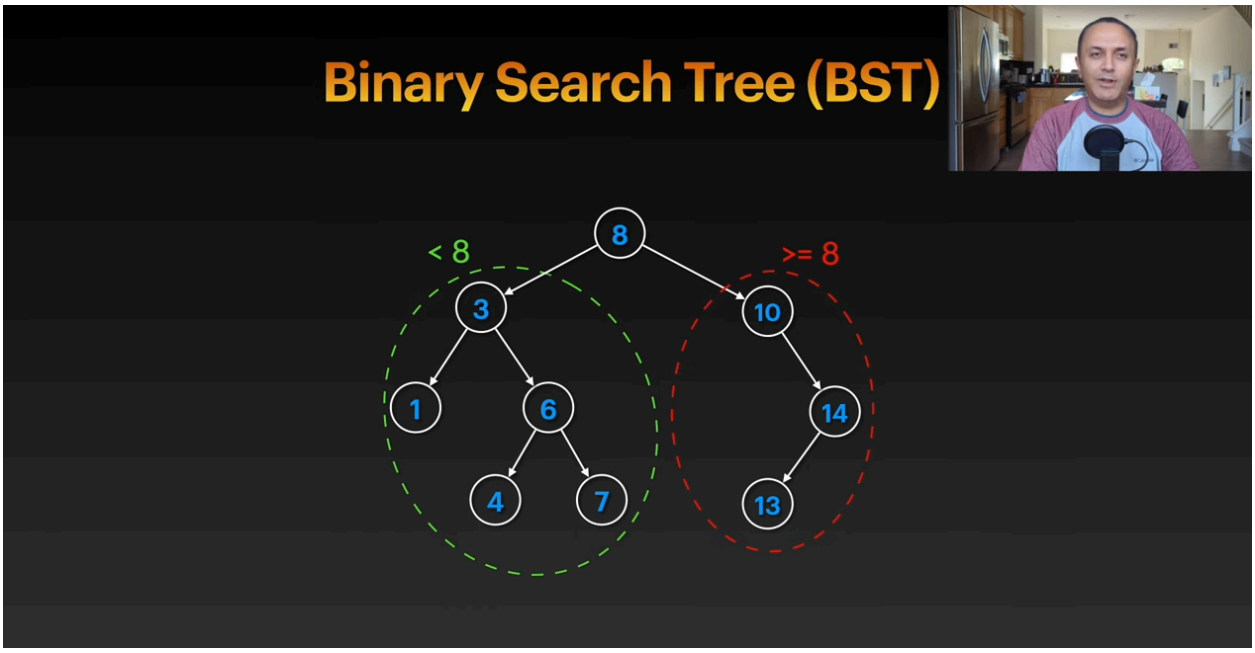
video#5

how to read data

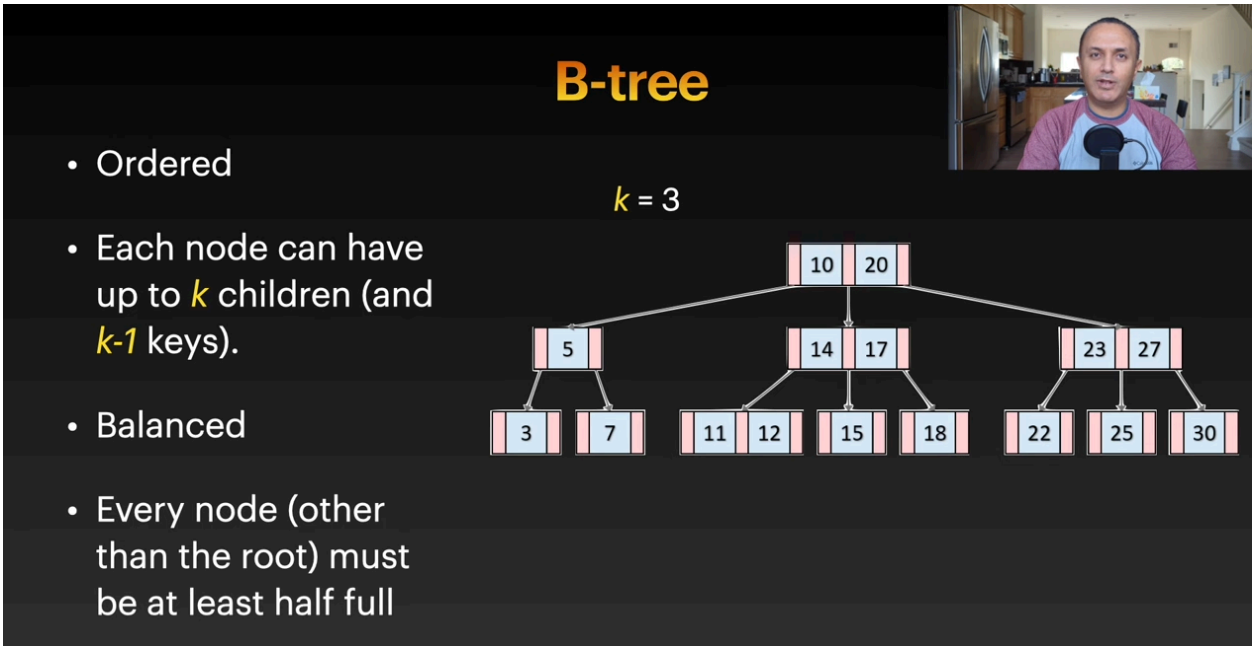
1.full table scan بنقري كل فايل و كل البيج ال فيه و ندور ع حاجه معينه ودا بياخد وقت كبير عشان نحسن:

⇒ parallel scan
⇒ partitioning
⇒ indexes

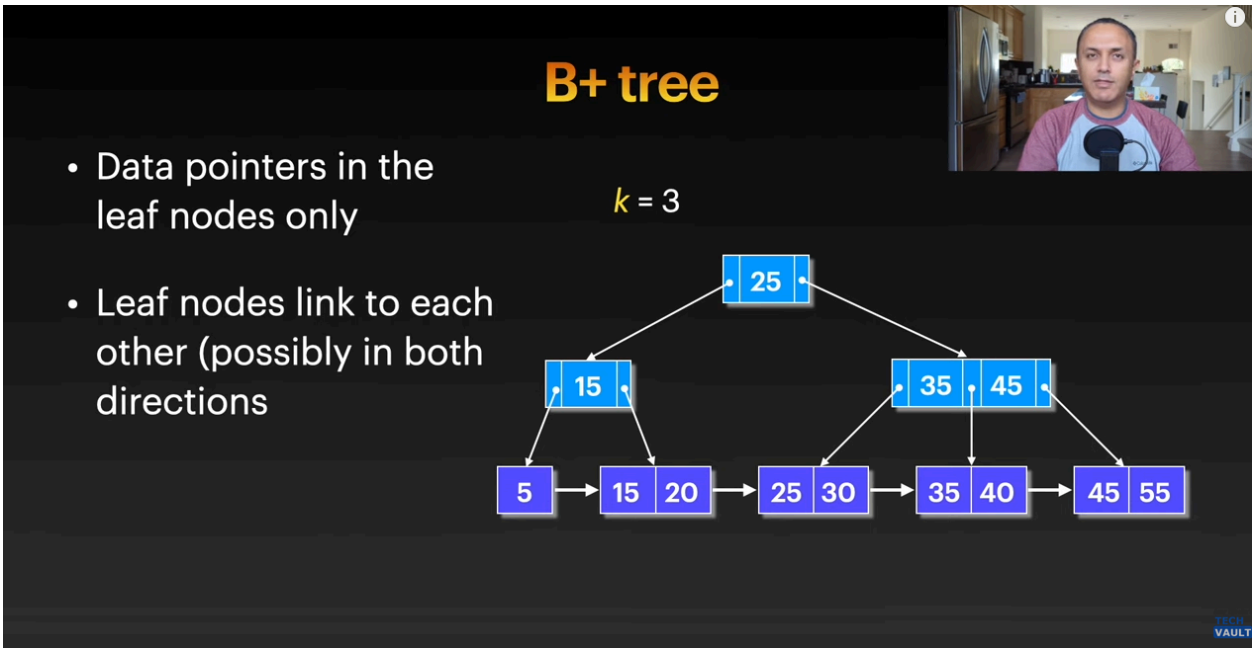
2. Binary search tree
⇒ Max two children.
⇒ Tree must be balanced ⇒ left sub tree and right sub tree are same height.



3. B-tree



4.B+tree

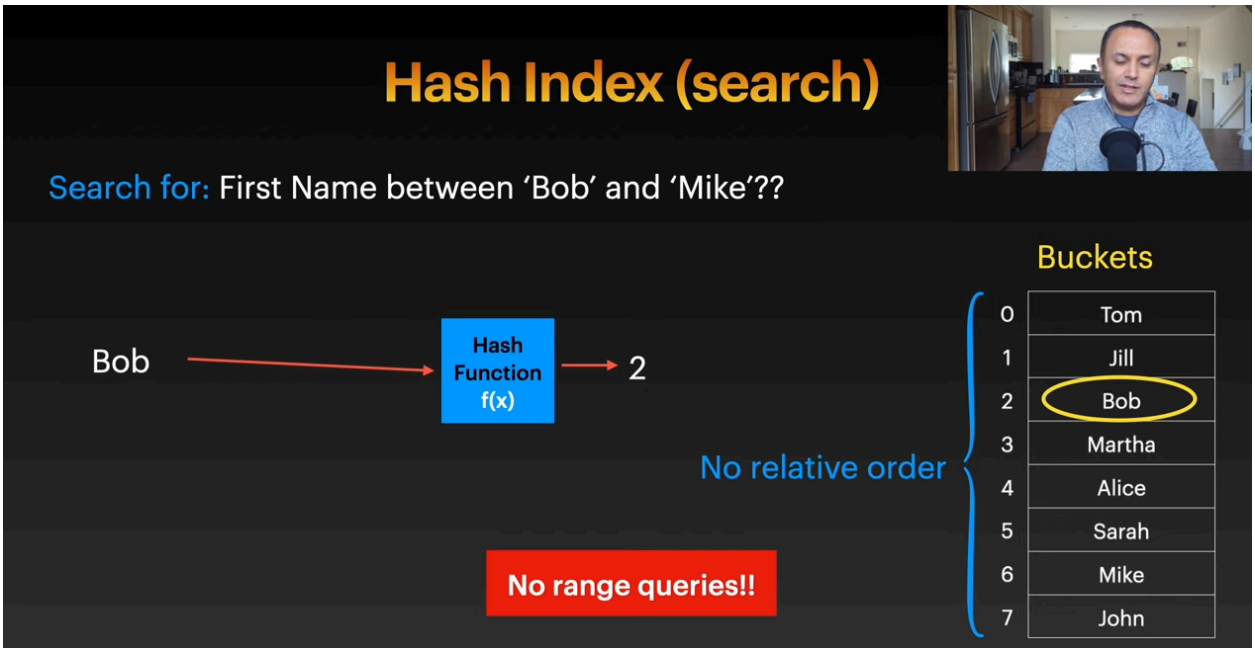


vido#6

Hash index ⇒ hash table (buckets)
⇒ hash function
Hash index faster than b-tree index

ب اختصار ال hash index متكون من buckets و hash function فا ال search key بتدخل في ال function بطلع رقم الخانة في ال buckets بيتخزن فيها القيمة

⇒ not ordered
⇒ so can't use in range query
⇒can use in point query



Composite index

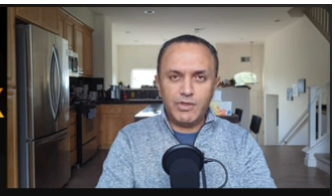
1⇒Composite hash index

⇒ function can take two input
⇒can't use in range query
⇒can use in point query

2⇒Composite b-tree index

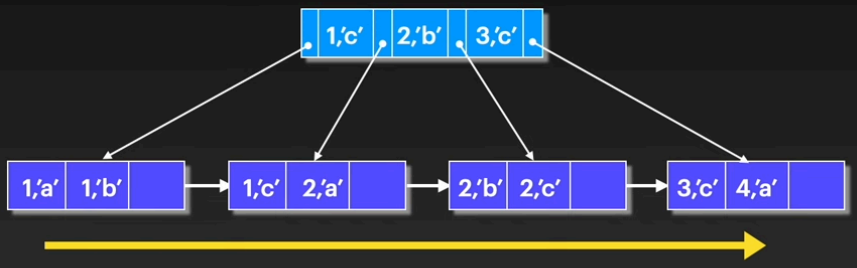
⇒ordered
⇒can use in range query

Composite B+ Tree Index



Index on attributes: x, y ≠ Index on attributes: y, x

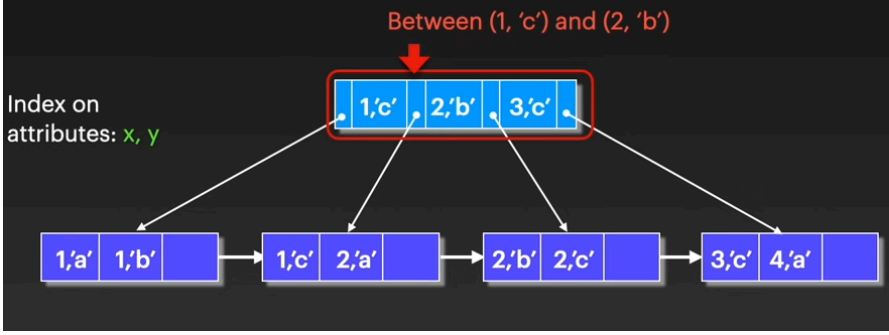
x	y
1	'a'
1	'b'
1	'c'
2	'a'
2	'b'
2	'c'
3	'c'
4	'a'



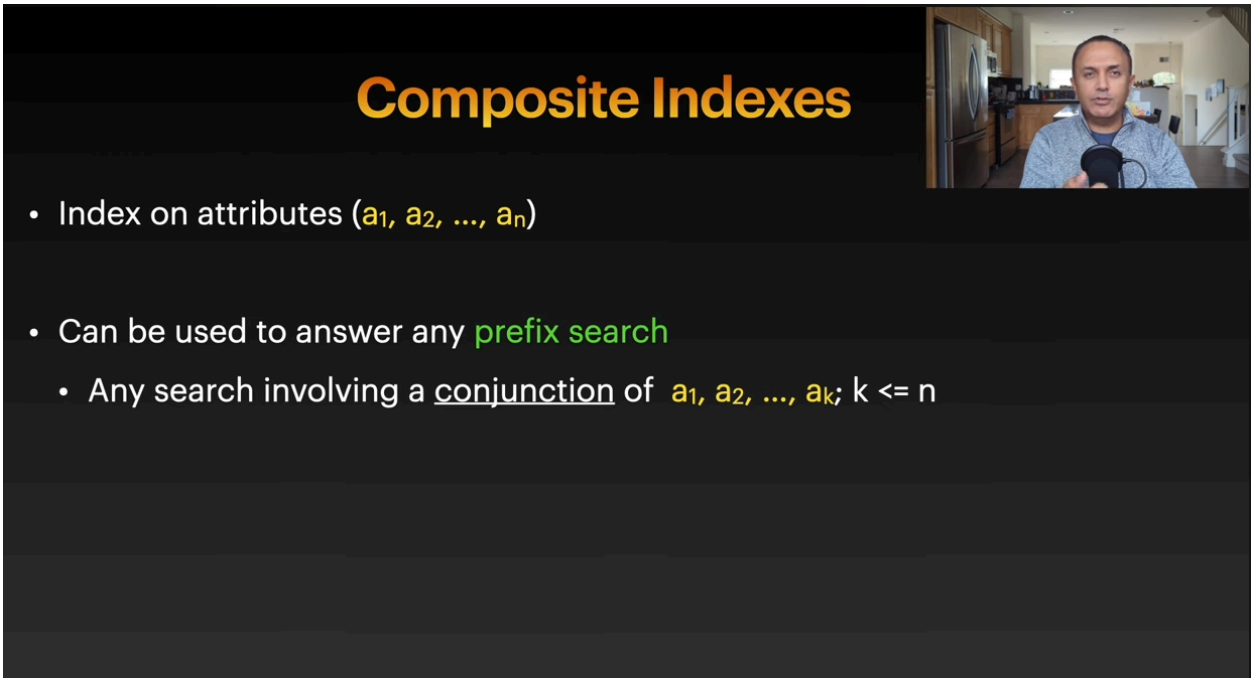
Query

Search for: x = 2 AND y = 'a'

Look for: (2, 'a') —> Point query



x	y
1	'a'
1	'b'
1	'c'
2	'a'
2	'b'
2	'c'
3	'c'
4	'a'



video#7

Problems of composite index

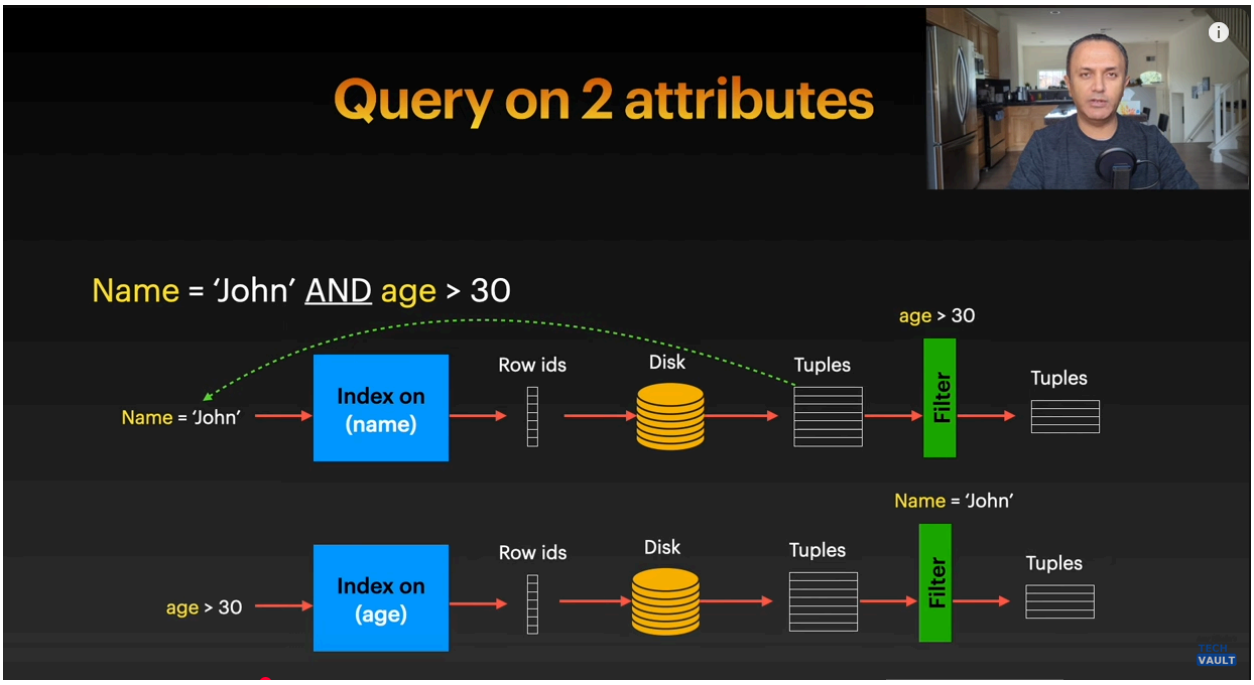
1.size

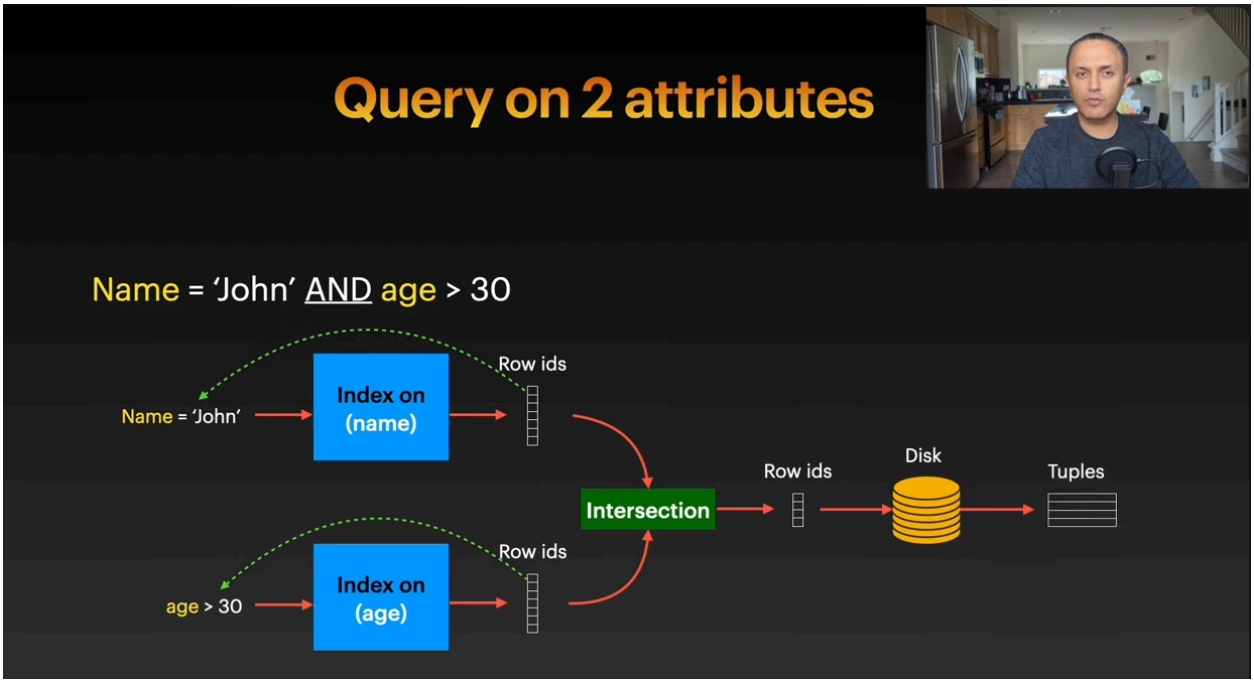
2.efficiency

3.complexity

4.too many options

5.disjunction (or)





Another way to use index and avoid this problems

Index intersection

⇒ use two indexes(more than one index)

Features

1.flexiblity

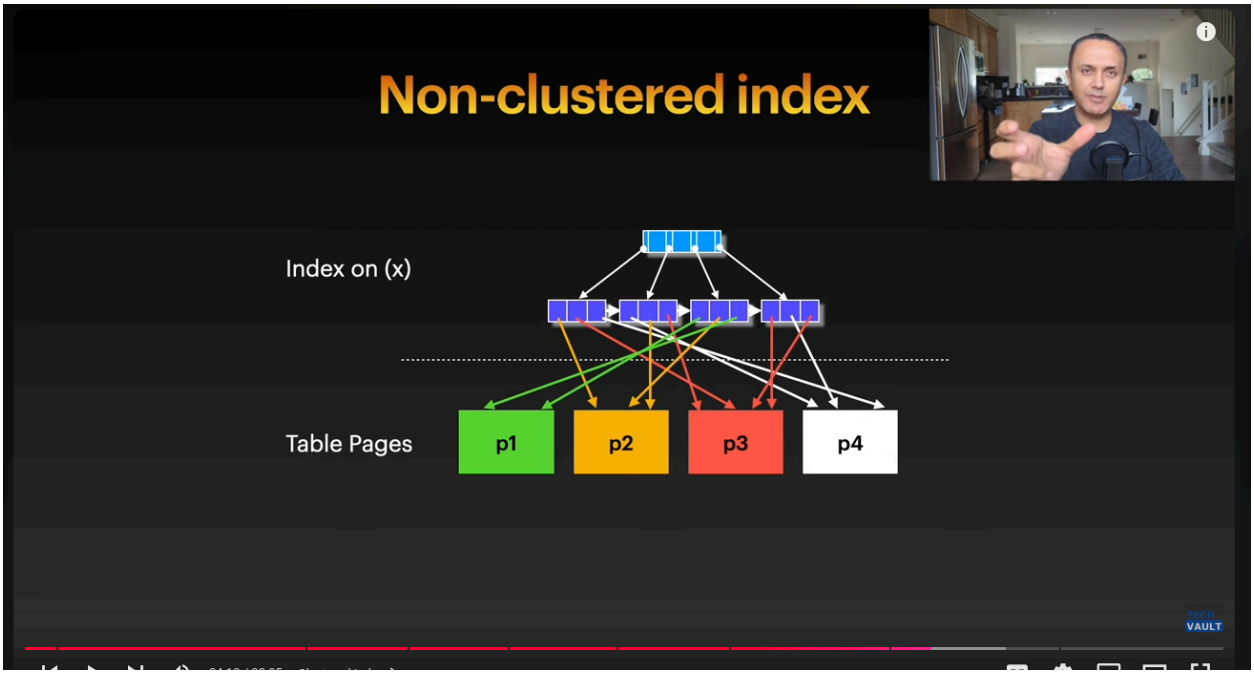
2.storage space

3.update cost
Non-features
1.execution cost
2.optimizer complexity

Covering index:

Non-clustered index

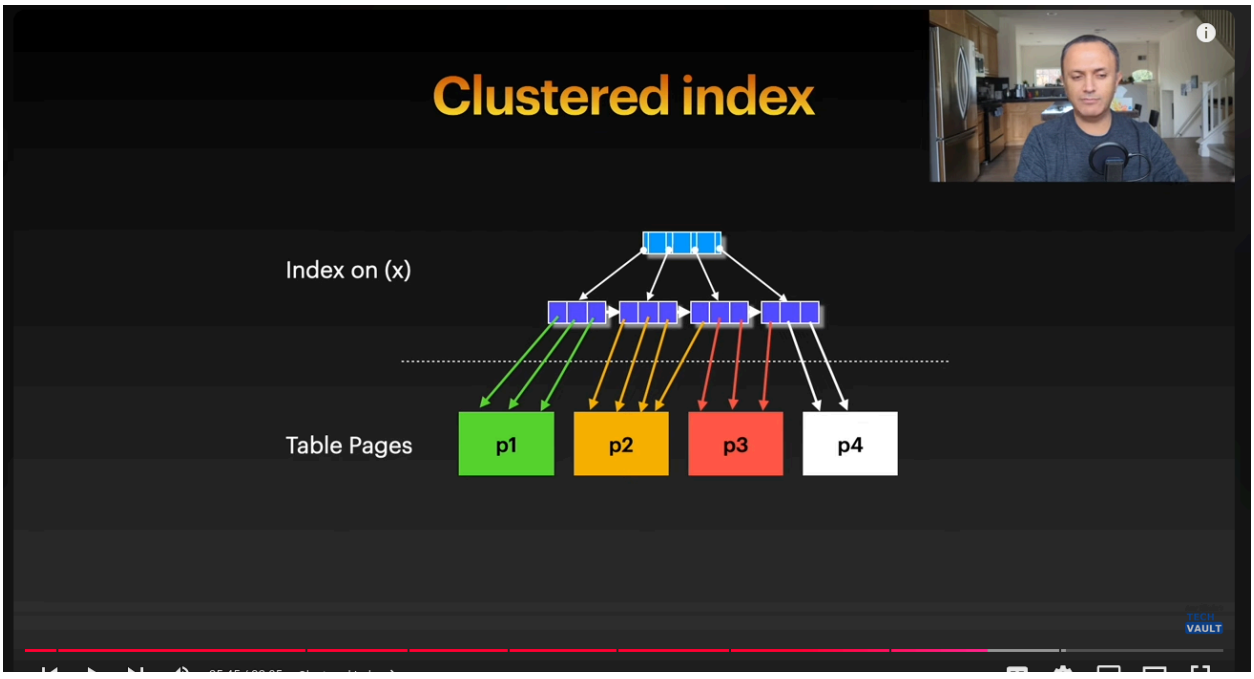
Index ordered
But pages not
B3ml too much random access



clustered index

Index ordered

pages ordered



video#9

Query lifecycle:
CRUD operation:

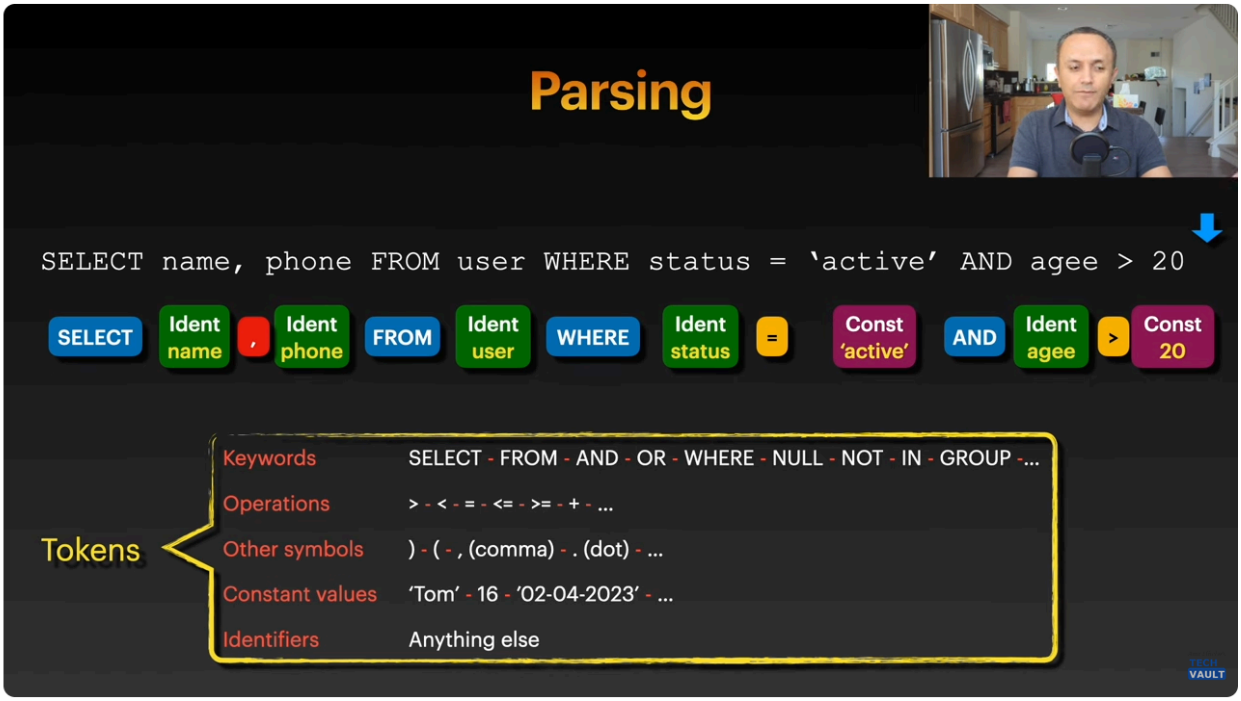
1.create / insert
2.read
3.update
4.delete

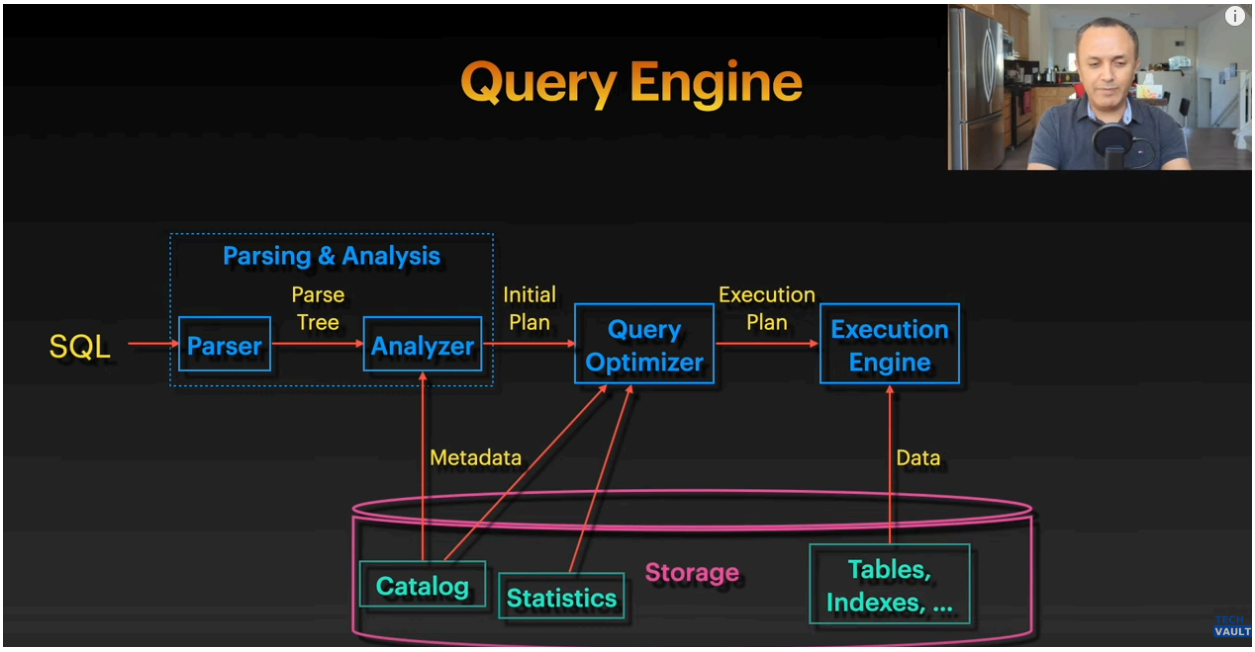
Engin understand tokens:
Tokens:
1.keyword (select , from ,and ,or)
2.operations (< , > , = , +)
3.other symbols ((,),, .)
4.constant value
5.identifers (col name , variables , table name)

Query lifecycle:

Sql Parsing ⇒ Analyzer ⇒ query optimizer ⇒execution engine.

1.Sql parsing / syntax checking ⇒ bt3ml check 3la tokens , **result** ⇒parse tree.
2.Analyzer ⇒ check 3la el identifiers by checking el catalog , **result** ⇒ initial plan.
3.query optimizer ⇒ more complex , know cost for each plan , **result** ⇒ execution plan.
4.execution engine.





==========

video#10

Join query

Join plan

Each input can be a subplan.

Left ⇒ (outer)
Right ⇒ (inner)

⇒ each join algorithm has a different cost.
⇒ cost can be estimated in terms of numbers of rows fetched from each side.
==========

Join algorithm

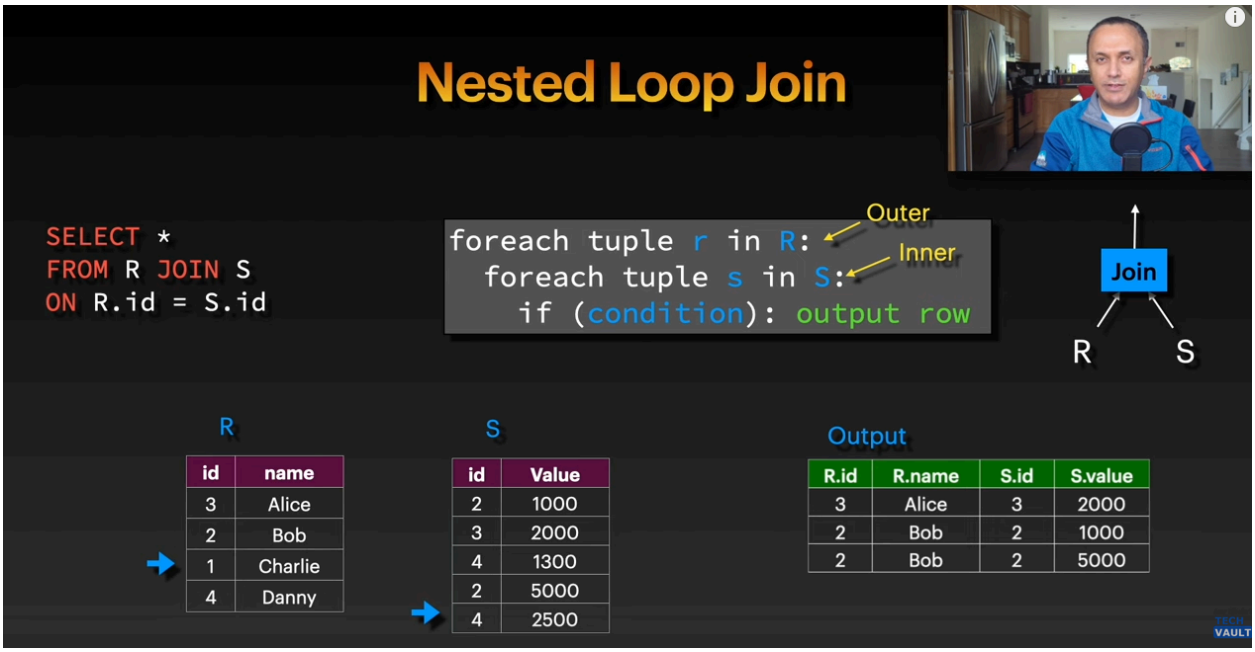
1. Nested loop join
2. Index nested loop join
3. **Merge join**
4. Hash join

==========

Query :
Select * from r join s on r.id = s.id

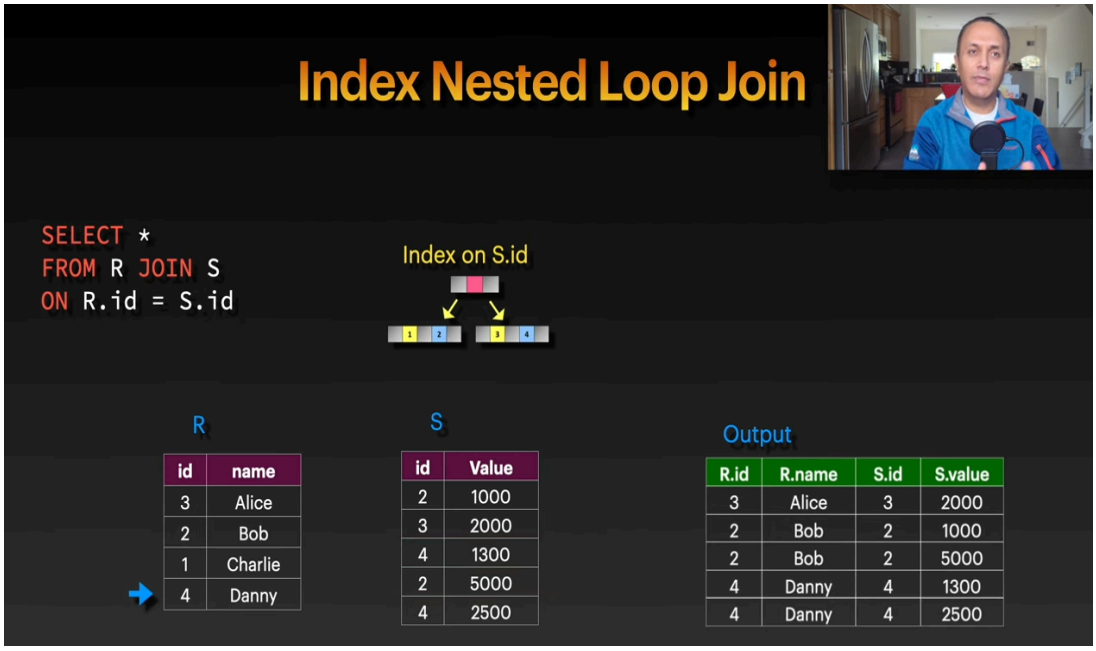
1.nested loop join

For each r in R :
For each tuple s in S
IF(r.id = s.id)
Cost = NR + (NR*NS) OR
NR + (NS * NR)
Nested loop full scan 3la inner table



2. Index nested loop join

Cost = Nouter + (N outer * C index)
We don't need full scan 3la inner table



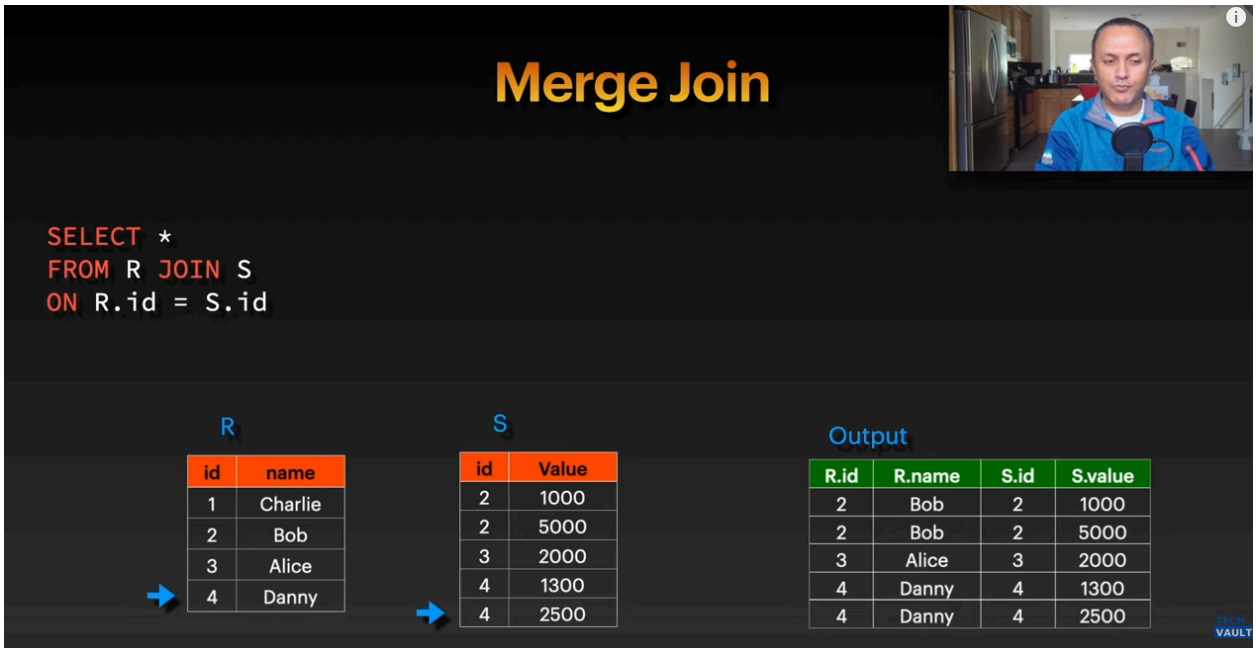
3.Merge join

Must be sorted.

Scan once for each table.

Used in equi - joins

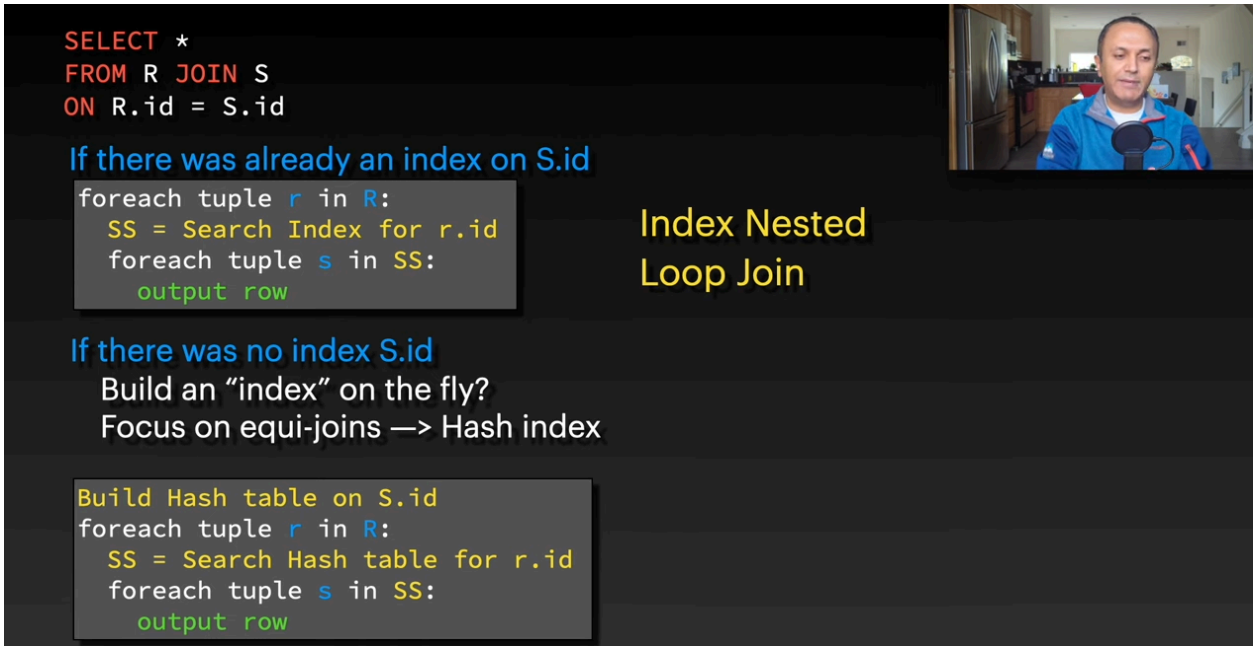
Cost = n outer + n inner + sort



4.Hash join

1. Built hash table
2. Lookup in hash table

Cost = (n outer + n inner)*c hash



==========

video#11

Aggregation

Combining and summarizing multiple data records into a single result set based on some criteria.

Aggregation steps:

1. Split input rows into groups .

Only if there are group by or distinct
Otherwise treat everything as a one big group.

2.combine value within each groups.
May use function count , min , max , sum , avg.

3.output one row per group.

==========
⇒ sort-base aggregation.
Must be sortted
⇒ hash-base aggregation.

==========

video#12

Optimizer

1.plan Enumeration

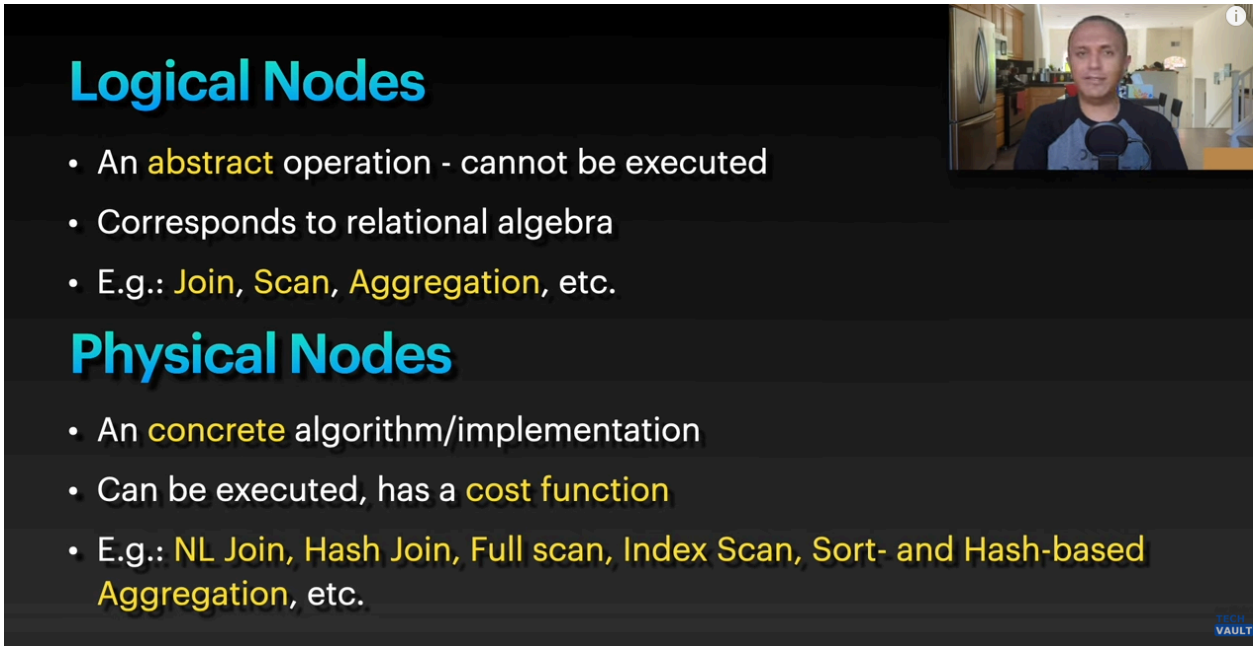
بيحاول يعرف ايه كل البلان الممكنه.
بيدرو ع كل البلان المكافئة للبلان الأساسية عندي وتطلعلي نفس النتيجة.

2.plan selection

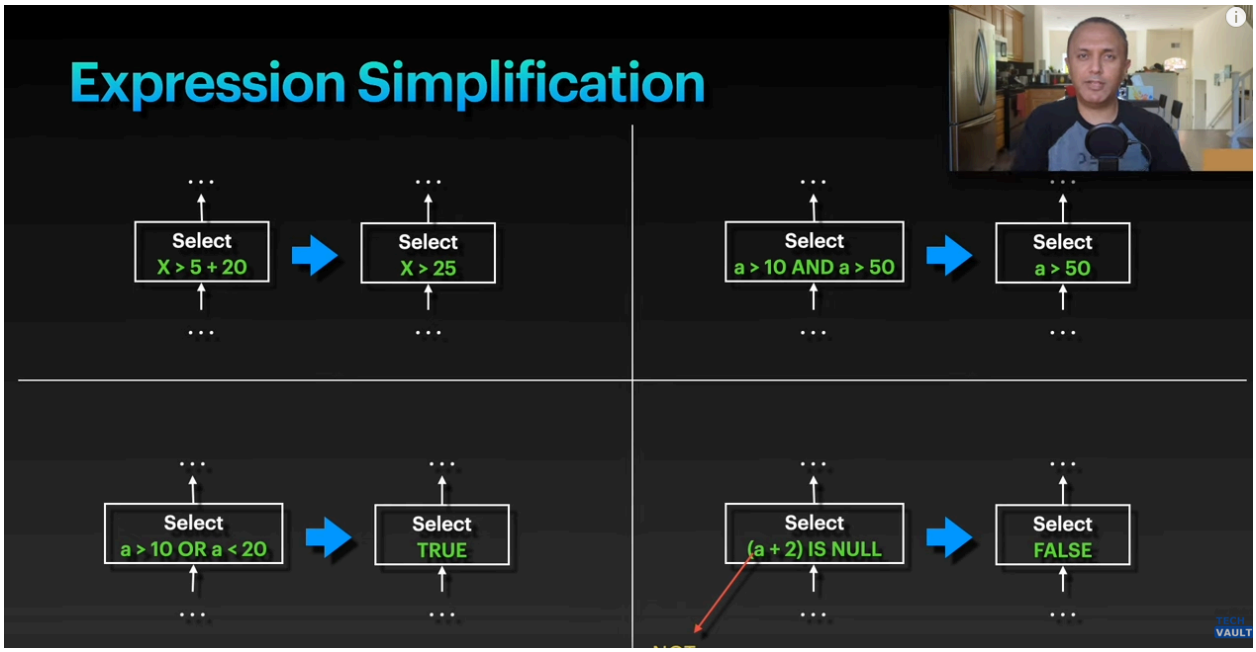
يقارن بينهم و يختار الاحسن
عن طريق:

.heuristics
.cost estimation

==========
Plan ⇒tree
Or directed acyclic graph
==========
plans⇒



Rule-based optimizati ⇒
Change from one plan to another
⇒ expression simplification



⇒ Cost-based optimization
Compare between old and new plan

Cost-based Optimization

- Join Order
- Access Path Selection (Full scan, index scan, etc.)
- Join algorithms (NL, Hash, Merge...)
- Aggregation algorithms
- Sorting algorithms
- Etc.

Bottom-Up Optimization

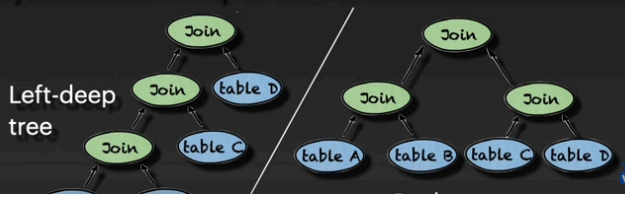
- For the given query, enumerate and compare sub-plans with:
  - 1 table
  - 2 tables
  - 3 tables
  - etc.
- At each level find the best sub-plan, and use in higher levels

Bottom-Up Optimization

(+) Simple to implement

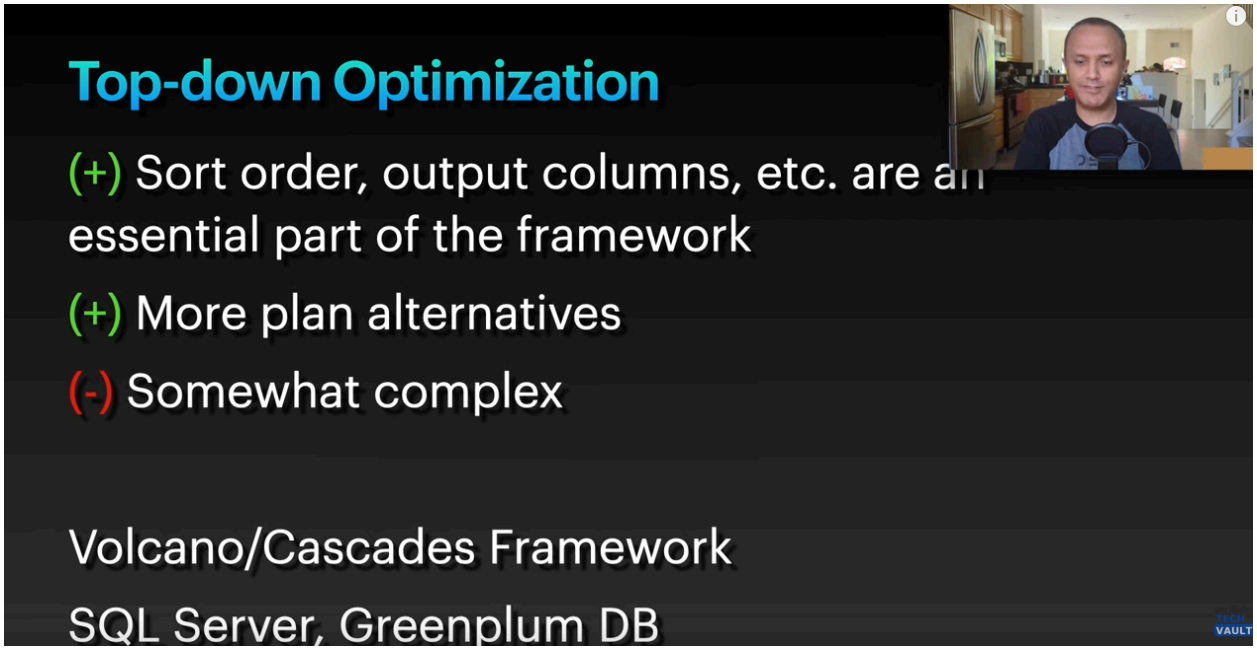
(-) Adding constraints about output column, sort order, etc. is not easy

(-) Usually considers only left-deep trees



Top-down Optimization

- Start from logical plan (after applying heuristic rules)
- Apply more rules to generate other logical and physical alternatives
- Estimate cost and choose cheapest alternative


Top-down Optimization
(+) Sort order, output columns, etc. are an
essential part of the framework
(+) More plan alternatives
(-) Somewhat complex
Volcano/Cascades Framework
SQL Server, Greenplum DB
VAULT