Database

26 January 2024 00:21

Problem with old Database

- Complex Queries
- Easily Corrupted

Then came hierarchical model
Problem
- Duplicate Data
- No Independence

Object-Oriented Model
Problem
Complex Queries
No Standard API

Scaling or Elasticity is the degree to which a system can adapt to workload changes by provisioning and deprovisioning resources in an on-demand manner

Rise of the special purpose OLAP DBMSs.
→Distributed / Shared-Nothing
→Relational / SQL
→Usually closed-source.

2000s NoSQL SYSTEMS
Focus on high-availability & high-scalability:
→ Schemaless (i.e., "Schema Last")
→ Non-relational data models (document, key/value, etc)
→ No ACID transactions
→ Custom APIs instead of SQL
→ Usually open-source

2010s NewSQL
Provide same performance for OLTP workloads as NoSQL DBMSs without giving up ACID:
→Relational / SQL
→Distributed
→Usually closed-source

2010s CLOUD SYSTEMS :- Computing environments delivering services over the internet.
Eg :- Amazon Aurora
2010s SHARED-DISK ENGINES :- Database or data processing systems with multiple nodes sharing access to a common disk storage.
Eg :- Amazon Redshift
2010s TIMESERIES SYSTEMS :- Specialized systems that are designed to store timeseries / event data.
Eg :- influxDB

2010s SPECIALIZED SYSTEMS :-

Embedded DBMSs
Multi-Model DBMSs
Blockchain DBMSs
Hardware Acceleration

Big Data are high-volume, high-velocity, and/or high-variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization

Any data that exceeds our current capability of processing can be regarded as "big"

Areas of Applications
Health and Well being
Policy making and public opinions

Characters of Big Data

- **Volume**
- **Velocity :-**refers to the speed at which data is processed
- **Variety** :- Variety includes different types of data
  - Structured
  - Unstructured
  - Semi structured
- **Veracity** :- is the quality or trustworthiness of the data

The five critical differences between SQL and NoSQL are:
SQL databases are relational, and NoSQL databases are non-relational.

SQL databases use structured query language (SQL) and have a predefined schema. NoSQL databases have dynamic schemas for unstructured data.

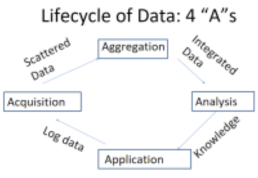
SQL databases are vertically scalable, while NoSQL databases are horizontally scalable.

SQL databases are table-based, while NoSQL databases are document, key-value, graph, or wide-column stores.

SQL databases are better for multi-row transactions, while NoSQL is better for unstructured data like documents or JSON.

TOP Open Source Big Data Databases

1. Cassandra Developed by facebook
2. HBase
3. MongoDB



The primary difference between relational and non-relational databases lies in their data storage and structure:

Relational Database (SQL):
1. **Structure**: Organized into tables with predefined columns and data types.
2. **Schema:** Follows a rigid, predefined schema.
3. **Scalability**: Generally vertically scalable, meaning it's scaled by increasing the power of the existing hardware.

Four characteristics of large data
4 'V's of Large Data – Volume, Velocity, Variety and Veracity.

Relational Databases:
- Use a structured schema with tables to organize and store data.
- Data is organized in rows and columns.
- Ensures data integrity through normalization.
- Typically use SQL (Structured Query Language) for querying.
- Well-suited for complex queries and transactions.

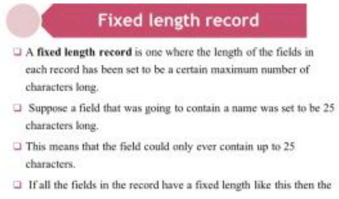
Non-Relational Databases:
- Use a flexible schema (schema-less) to store data.
- Data can be stored in various formats like key-value pairs, documents, graphs, or wide-column stores.
- Scalable and suitable for handling large volumes of unstructured or semi-structured data.
- No strict relationships between data entities.
- Common types include document-oriented databases, key-value stores, graph databases, and wide-column stores.

-NoSQL databases, designed for horizontal scalability, can efficiently handle large amounts of data and traffic, ensuring the system can scale as the game gains popularity.

-NoSQL databases are known for their ability to deliver fast read and write operations

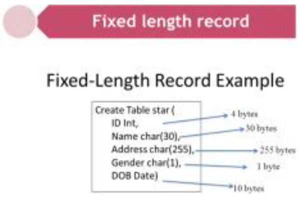
- NoSQL databases are often built with a focus on high availability and fault tolerance.

a Query is a request for information from a database. It is a command or a set of commands written in a specific language (often SQL - Structured Query Language) that is used to interact with a database and retrieve, modify, or manipulate data



Fixed length record

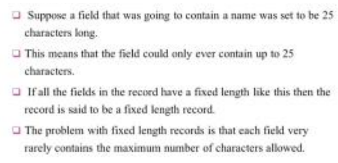
- ❑ A **fixed length record** is one where the length of the fields in each record has been set to be a certain maximum number of characters long.
- ❑ Suppose a field that was going to contain a name was set to be 25 characters long.
- ❑ This means that the field could only ever contain up to 25 characters.
- ❑ If all the fields in the record have a fixed length like this then the



Fixed length record

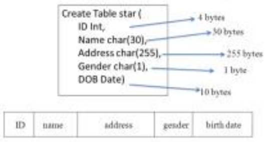
Fixed-Length Record Example

```
Create Table star (
    ID Int,
    Name char(30),
    Address char(255),
    Gender char(1),
    DOB Date)
```



- ❑ Suppose a field that was going to contain a name was set to be 25 characters long.
- ❑ This means that the field could only ever contain up to 25 characters.
- ❑ If all the fields in the record have a fixed length like this then the record is said to be a fixed length record.
- ❑ The problem with fixed length records is that each field very rarely contains the maximum number of characters allowed.

tai
sto
tru
mo
tru
of



```
Create Table star (
    ID Int,
    Name char(30),
    Address char(255),
    Gender char(1),
    DOB Date)
```

ID	name	address	gender	birth date

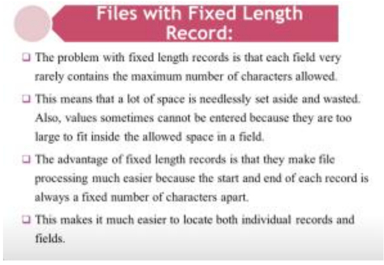
JOINS

A JOIN is a means for combining fields from two tables by using values common to each (in most of the cases by using foreign key).

(SQL) Joins can be classified into the following categories :

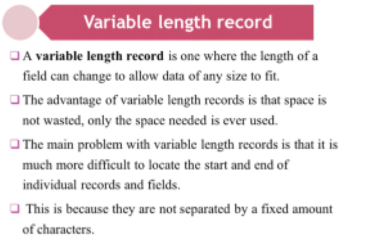
- **Cartesian Products**
- **Inner Joins (Equijoins)**
- **Self Joins**
- **Outer Joins (Left, Right and Full)**

Next slide



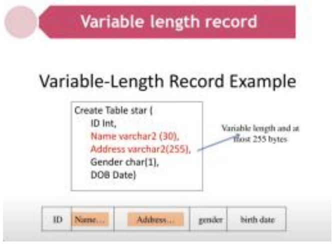
Files with Fixed Length Record:

- ❑ The problem with fixed length records is that each field very rarely contains the maximum number of characters allowed.
- ❑ This means that a lot of space is needlessly set aside and wasted. Also, values sometimes cannot be entered because they are too large to fit inside the allowed space in a field.
- ❑ The advantage of fixed length records is that they make file processing much easier because the start and end of each record is always a fixed number of characters apart.
- ❑ This makes it much easier to locate both individual records and fields.



Variable length record

- ❑ A **variable length record** is one where the length of a field can change to allow data of any size to fit.
- ❑ The advantage of variable length records is that space is not wasted, only the space needed is ever used.
- ❑ The main problem with variable length records is that it is much more difficult to locate the start and end of individual records and fields.
- ❑ This is because they are not separated by a fixed amount of characters.



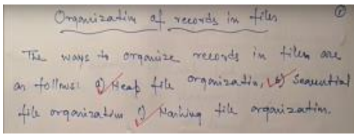
Variable length record

Variable-Length Record Example

```
Create Table star (
    ID Int,
    Name varchar2 (30),
    Address varchar2(255),
    Gender char(1),
    DOB Date)
```

ID	Name…	Address…	gender	birth date

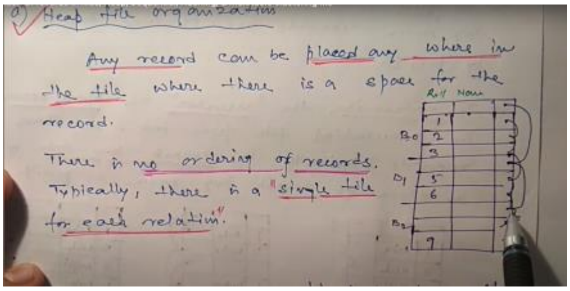
Organization of Records in Files
Yeh Notebook mein hai



Organization of records in files

The ways to organize records in files are as follows: a) Heap file organization, b) Sequential file organization c) Hashing file organization.

Heap – a record can be placed anywhere in the file where there is space



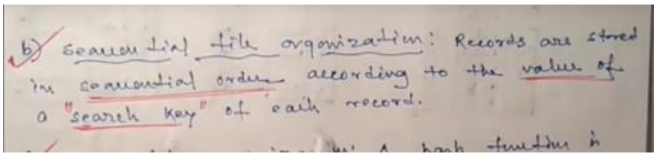
a) Heap file organization

Any record can be placed any where in the file where there is a space for the record.

There is no ordering of records. Typically, there is a "single file for each relation".

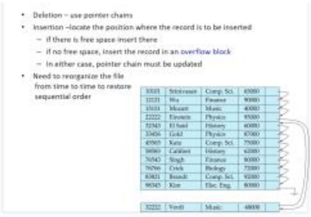
Roll Nos
B0
1
2
3
B1
5
6
B2
7

Sequential – store records in sequential order, based on the value of the search key of each record
The records in the file are ordered by a **search-key**



b) Sequential file organization: Records are stored in sequential order according to the value of a "search key" of each record.

A hash function is



- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order

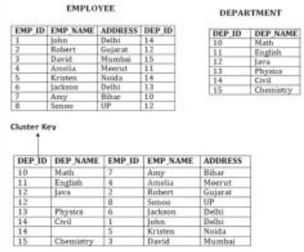
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

32222	Verdi	Music	48000

Hashing – a hash function computed on some attribute of each record; the result specifies in which block of the file the record should be placed

Multitable Clustering File Organization

Multitable clustering refers to a database design approach where multiple tables are organized or clustered together based on certain criteria. It aims to optimize data retrieval and improve overall database efficiency.



EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEP_ID
1	John	Delhi	14
2	Robert	Gujarat	12
3	David	Mumbai	15
4	Amelia	Meerut	11
5	Kristen	Noida	14
6	Jackson	Delhi	13
7	Amy	Bihar	10
8	Sonoo	UP	12

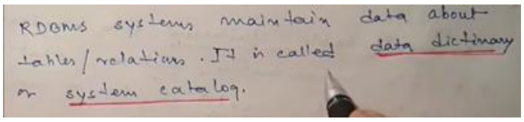
DEPARTMENT

DEP_ID	DEP_NAME
10	Math
11	English
12	Java
13	Physics
14	Civil
15	Chemistry

Cluster Key

DEP_ID	DEP_NAME	EMP_ID	EMP_NAME	ADDRESS
10	Math	7	Amy	Bihar
11	English	4	Amelia	Meerut
12	Java	2	Robert	Gujarat
12		8	Sonoo	UP
13	Physics	6	Jackson	Delhi
14	Civil	1	John	Delhi
14		5	Kristen	Noida
15	Chemistry	3	David	Mumbai

Data Dictionary Storage

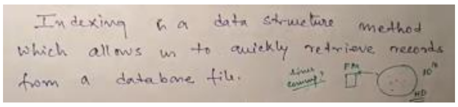


RDBMS systems maintain data about tables/relations. It is called data dictionary or system catalog.

Storage Access (YEH TOPIC REH GYA HAI)

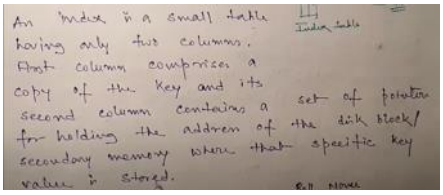
Indexing

Data is present in the hard drive so we have to first search for the data in hard drive and then bring that data in primary memory to retrieve it.
So it is a very time consuming process



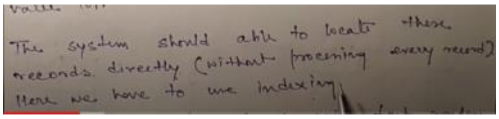
Indexing is a data structure method which allows us to quickly retrieve records from a database file.

time consuming?
PM
10^n
HD

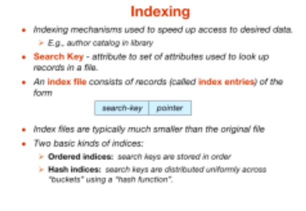


An index is a small table having only two columns. First column comprises a copy of the key and its second column contains a set of pointers for holding the address of the disk block/ secondary memory where that specific key value is stored.

Index table
Roll Name



The system should able to locate these records directly (without processing every record) Here we have to use indexing

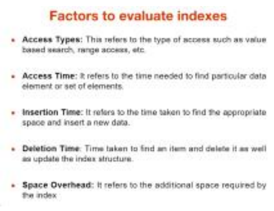


Indexing

- *Indexing mechanisms used to speed up access to desired data.*
  - *E.g., author catalog in library*
- **Search Key** - *attribute to set of attributes used to look up records in a file.*
- *An* **index file** *consists of records (called* **index entries**) *of the form*

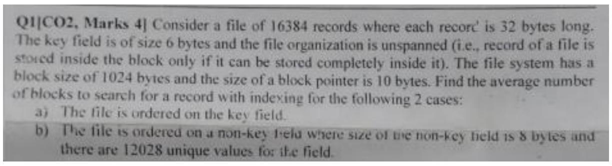
search-key	pointer

- *Index files are typically much smaller than the original file*
- *Two basic kinds of indices:*
  - **Ordered indices:** *search keys are stored in order*
  - **Hash indices:** *search keys are distributed uniformly across "buckets" using a "hash function".*



Factors to evaluate indexes

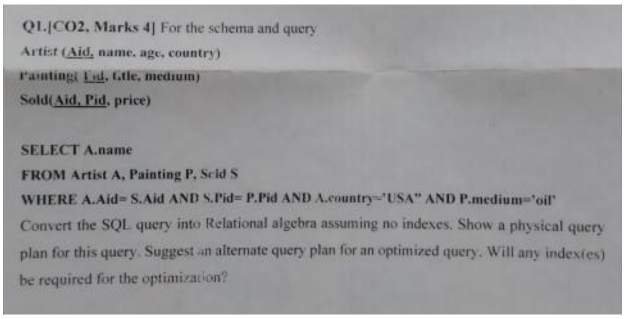
- **Access Types:** This refers to the type of access such as value based search, range access, etc.
- **Access Time:** It refers to the time needed to find particular data element or set of elements.
- **Insertion Time:** It refers to the time taken to find the appropriate space and insert a new data.
- **Deletion Time**: Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead:** It refers to the additional space required by the index



Q1[CO2, Marks 4] Consider a file of 16384 records where each recorc' is 32 bytes long. The key field is of size 6 bytes and the file organization is unspanned (i.e., record of a file is stored inside the block only if it can be stored completely inside it). The file system has a block size of 1024 bytes and the size of a block pointer is 10 bytes. Find the average number of blocks to search for a record with indexing for the following 2 cases:

a) The file is ordered on the key field.

b) The file is ordered on a non-key field where size of the non-key field is 8 bytes and there are 12028 unique values for the field.



Q1.[CO2, Marks 4] For the schema and query

Artist (Aid, name, age, country)

Painting(Pid, title, medium)

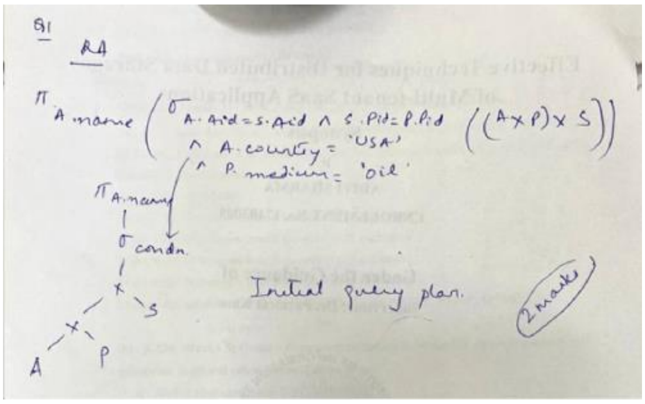
Sold(Aid, Pid, price)

SELECT A.name

FROM Artist A, Painting P, Sold S

WHERE A.Aid= S.Aid AND S.Pid= P.Pid AND A.country='USA" AND P.medium='oil'

Convert the SQL query into Relational algebra assuming no indexes. Show a physical query plan for this query. Suggest an alternate query plan for an optimized query. Will any index(es) be required for the optimization?



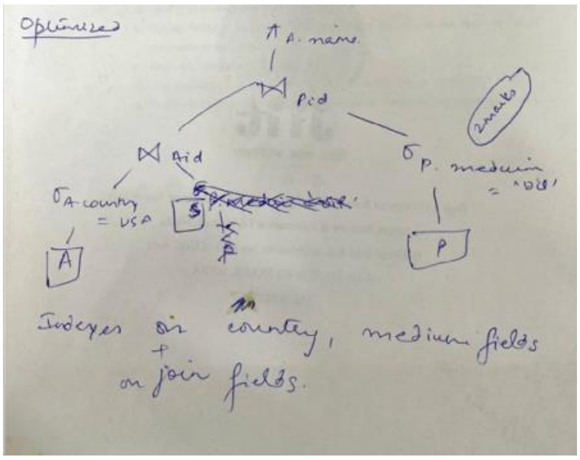
Q1

RA

$$\pi_{A.name}\left(\sigma_{A.Aid=S.Aid \wedge S.Pid=P.Pid \wedge A.country='USA' \wedge P.medium='oil'}\left(\left(A \times P\right) \times S\right)\right)$$

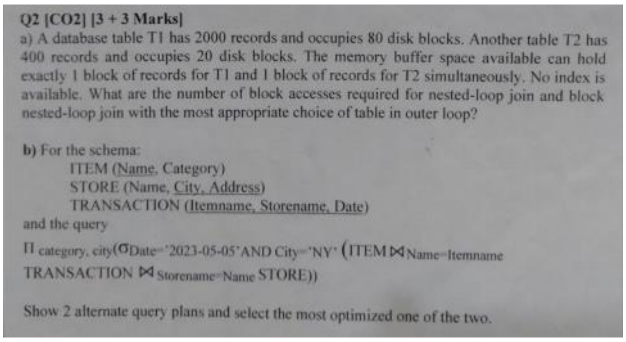
Initial query plan.

2 marks



Optimized

Indexes on country, medium fields + on join fields.



Q2 [CO2] [3 + 3 Marks]

a) A database table T1 has 2000 records and occupies 80 disk blocks. Another table T2 has 400 records and occupies 20 disk blocks. The memory buffer space available can hold exactly 1 block of records for T1 and 1 block of records for T2 simultaneously. No index is available. What are the number of block accesses required for nested-loop join and block nested-loop join with the most appropriate choice of table in outer loop?

b) For the schema:

ITEM (Name, Category)
STORE (Name, City, Address)
TRANSACTION (Itemname, Storename, Date)

and the query

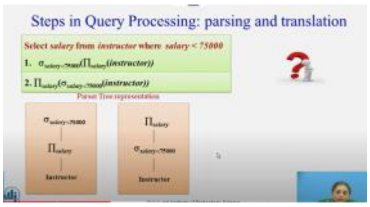
$$\Pi_{category,\ city}(\sigma_{Date='2023-05-05' AND\ City='NY'}(ITEM \bowtie_{Name=Itemname} TRANSACTION \bowtie_{Storename=Name} STORE))$$

Show 2 alternate query plans and select the most optimized one of the two.

neeche se chalta hia

signa no of row kam karta hai
pie no of columne ko kum kar deta

cross join



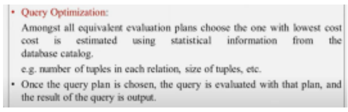
Steps in Query Processing: parsing and translation

Select salary from instructor where salary < 75000

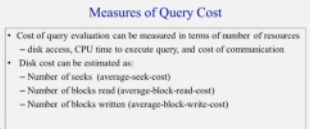
1. $\sigma_{salary<75000}(\Pi_{salary}(instructor))$

2. $\Pi_{salary}(\sigma_{salary<75000}(instructor))$

Parse Tree representation

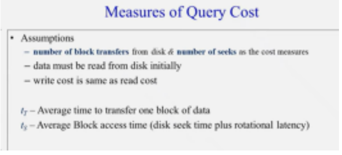


- Query Optimization:
  Amongst all equivalent evaluation plans choose the one with lowest cost cost is estimated using statistical information from the database catalog.
  e.g. number of tuples in each relation, size of tuples, etc.
- Once the query plan is chosen, the query is evaluated with that plan, and the result of the query is output.



Measures of Query Cost

- Cost of query evaluation can be measured in terms of number of resources
  - disk access, CPU time to execute query, and cost of communication
- Disk cost can be estimated as:
  - Number of seeks (average-seek-cost)
  - Number of blocks read (average-block-read-cost)
  - Number of blocks written (average-block-write-cost)

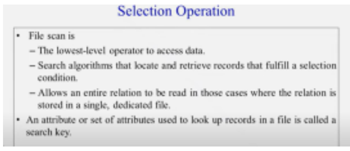


Measures of Query Cost

- Assumptions
  - **number of block transfers** from disk & **number of seeks** as the cost measures
  - data must be read from disk initially
  - write cost is same as read cost

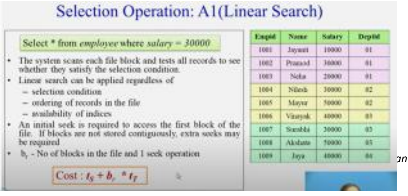
t_T – Average time to transfer one block of data
t_S – Average Block access time (disk seek time plus rotational latency)

Selection Operation



Selection Operation

- File scan is
  - The lowest-level operator to access data.
  - Search algorithms that locate and retrieve records that fulfill a selection condition.
  - Allows an entire relation to be read in those cases where the relation is stored in a single, dedicated file.
- An attribute or set of attributes used to look up records in a file is called a search key.



Selection Operation: A1(Linear Search)

Select * from *employee* where *salary* = *30000*

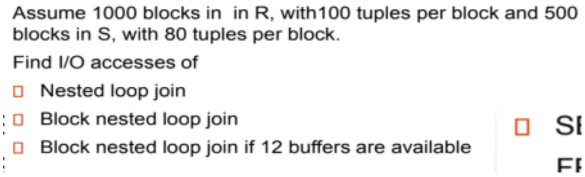
- The system scans each file block and tests all records to see whether they satisfy the selection condition.
- Linear search can be applied regardless of
  - selection condition
  - ordering of records in the file
  - availability of indices
- An initial seek is required to access the first block of the file. If blocks are not stored contiguously, extra seeks may be required
- b_r - No of blocks in the file and 1 seek operation

Cost : $t_S + b_r * t_T$

Empid	Name	Salary	Deptid
1001	Jayant	10000	01
1002	Pramod	30000	01
1003	Neha	20000	01
1004	Nilesh	30000	02
1005	Mayur	50000	02
1006	Vinayak	40000	03
1007	Surabhi	30000	03
1008	Akshata	50000	03
1009	Jaya	40000	04

SQL In
True:
INSER
is a tr
ance

Assume 1000 blocks in in R, with100 tuples per block and 500 blocks in S, with 80 tuples per block.
Find I/O accesses of
Nested loop join
Block nested loop join
Block nested loop join if 12 buffers are available

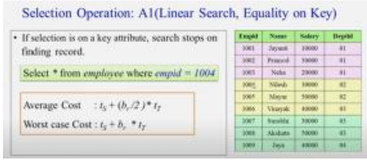


Assume 1000 blocks in in R, with100 tuples per block and 500 blocks in S, with 80 tuples per block.

Find I/O accesses of

- □ Nested loop join
- □ Block nested loop join
- □ Block nested loop join if 12 buffers are available

□ SI
FI



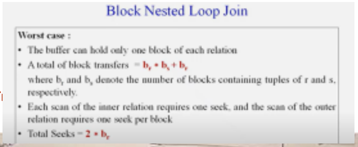
Selection Operation: A1(Linear Search, Equality on Key)

- If selection is on a key attribute, search stops on finding record.

Select * from *employee* where *empid* = *1004*

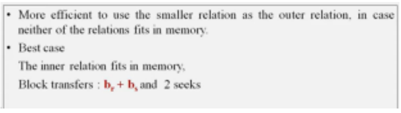
Average Cost : $t_S + (b_r/2) * t_T$
Worst case Cost : $t_S + b_r * t_T$

Empid	Name	Salary	Deptid
1001	Jayant	10000	01
1002	Pramod	30000	01
1003	Neha	20000	01
1004	Nilesh	30000	02
1005	Mayur	50000	02
1006	Vinayak	40000	03
1007	Surabhi	30000	03
1008	Akshata	50000	03
1009	Jaya	40000	04



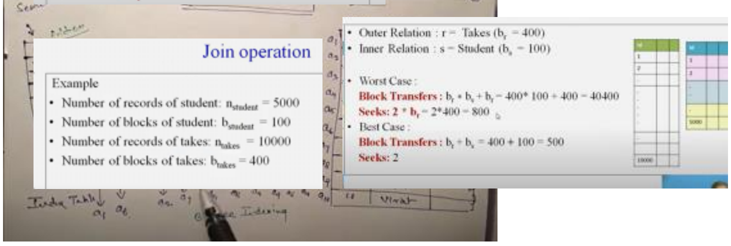
Block Nested Loop Join

Worst case :
- The buffer can hold only one block of each relation
- A total of block transfers = $b_r * b_s + b_r$
  where b_r and b_s denote the number of blocks containing tuples of r and s, respectively.
- Each scan of the inner relation requires one seek, and the scan of the outer relation requires one seek per block
- Total Seeks = $2 * b_r$



- More efficient to use the smaller relation as the outer relation, in case neither of the relations fits in memory.
- Best case
  The inner relation fits in memory.
  Block transfers : $b_r + b_s$ and 2 seeks

B+-T



Index Table
Virat

Join operation

Example
- Number of records of student: $n_{student}$ = 5000
- Number of blocks of student: $b_{student}$ = 100
- Number of records of takes: n_{takes} = 10000
- Number of blocks of takes: b_{takes} = 400

- Outer Relation : r = Takes (b_r = 400)
- Inner Relation : s = Student (b_s = 100)
- Worst Case :
  Block Transfers : $b_r * b_s + b_r = 400 * 100 + 400 = 40400$
  Seeks: $2 * b_r = 2*400 = 800$
- Best Case :
  Block Transfers : $b_r + b_s = 400 + 100 = 500$
  Seeks: 2

Disadvantage of indexed-sequential files
- performance degrades as file grows, since many overflow blocks get created.
- Periodic reorganization of entire file is required.

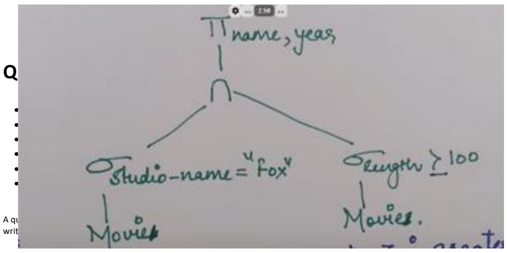
Advantage of B+-tree index files:
- automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
- Reorganization of entire file is not required to maintain performance.

A B⁺-tree is a rooted tree satisfying the following properties:

- *All paths from root to leaf are of the same length*
- *Each node that is not a root or a leaf has between ⌈n/2⌉ and n children.*
- *A leaf node has between ⌈(n–1)/2⌉ and n–1 values*
- *Special cases:*
  - *If the root is not a leaf, it has at least 2 children.*
  - *If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and (n–1) values.*

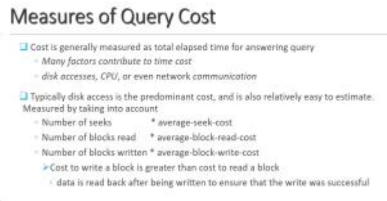
Q

A qu
writ



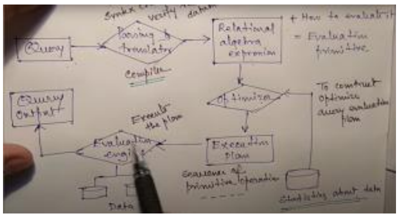
Basic Steps in Query Processing

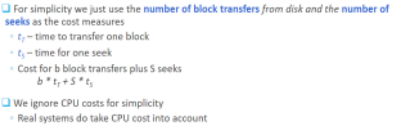
- **Parsing and translation**
  translate the query into its internal form. This is then translated into relational algebra.
  Parser checks syntax, verifies relations
- **Optimization**
  Amongst all equivalent evaluation plans choose the one with lowest cost.
  Cost is estimated using statistical information from the database catalog
  e.g. number of tuples in each relation, size of tuples, etc.
- **Evaluation**
  The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.



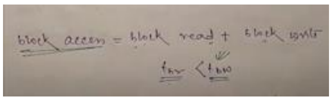
Measures of Query Cost

- Cost is generally measured as total elapsed time for answering query
  - *Many factors contribute to time cost*
  - *disk accesses, CPU, or even network communication*
- Typically disk access is the predominant cost, and is also relatively easy to estimate. Measured by taking into account
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost
    - Cost to write a block is greater than cost to read a block
      - data is read back after being written to ensure that the write was successful





- ❑ For simplicity we just use the **number of block transfers** *from disk and the* **number of seeks** as the cost measures
  - t_T – time to transfer one block
  - t_S – time for one seek
  - Cost for b block transfers plus S seeks
    $b * t_T + S * t_S$
- ❑ We ignore CPU costs for simplicity
  - Real systems do take CPU cost into account

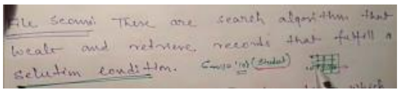


block access = block read + block write

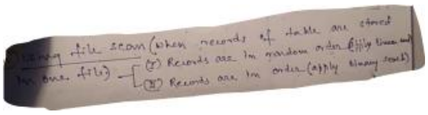
$t_{br} < t_{bw}$

Block read time is less than block write time

Several algorithms can reduce disk IO by using extra buffer space



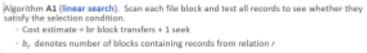
File Scan: These are search algorithm that locate and retrieve records that fulfill a selection condition. $\sigma_{ID='10'}$(Student)



...ing file scan (when records of table are stored in one file) — (I) Records are in random order (apply linear search)
(II) Records are in order (apply binary search)

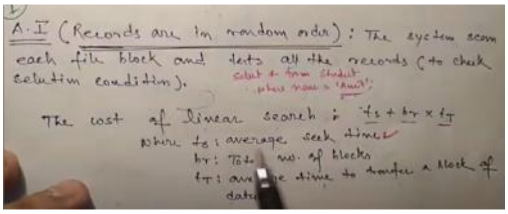
Selection Operation

File scan



Algorithm **A1** (**linear search**). Scan each file block and test all records to see whether they satisfy the selection condition.

- Cost estimate = br block transfers + 1 seek
- b_r denotes number of blocks containing records from relation *r*



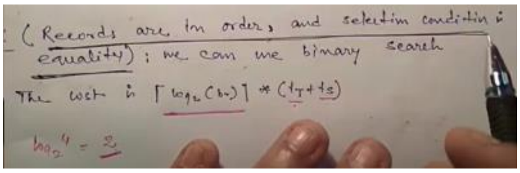
A.I (Records are in random order): The system scan each file block and tests all the records (to check selection condition).

select * from Student
where name = 'Amit';

The cost of linear search: $t_s + b_r \times t_T$

where t_s: average seek time
b_r: Tota... no. of blocks
t_T: ave...e time to transfer a block of data

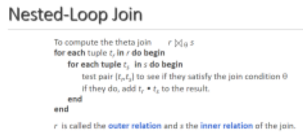
Binary search Applicable if selection is an equality comparison on the attribute on which file is ordered and when there is an index available, comes under Algorithm A2



(Records are in order, and selection condition is equality): we can use binary search

The cost is $\lceil \log_2(b_r) \rceil * (t_T + t_S)$

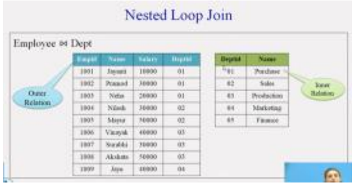
$\log_2 4 = 2$



Nested-Loop Join

```
To compute the theta join   $r \bowtie_\theta s$
for each tuple $t_r$ in $r$ do begin
    for each tuple $t_s$ in $s$ do begin
        test pair $(t_r, t_s)$ to see if they satisfy the join condition $\theta$
        if they do, add $t_r \cdot t_s$ to the result.
    end
end
```

r is called the **outer relation** and *s* the **inner relation** of the join.

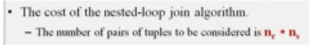


Nested Loop Join

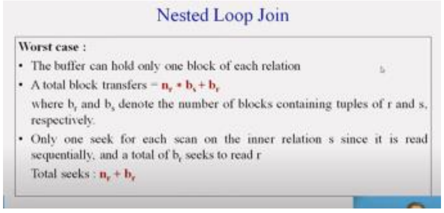
Employee ⋈ Dept

EmpId	Name	Salary	DeptId
1001	Jayanti	10000	01
1002	Pramod	30000	01
1003	Neha	20000	01
1004	Nilesh	30000	02
1005	Mayur	50000	02
1006	Vinayak	40000	03
1007	Surabhi	30000	03
1008	Akshata	50000	03
1009	Jaya	40000	04

DeptId	Name
01	Purchase
02	Sales
03	Production
04	Marketing
05	Finance



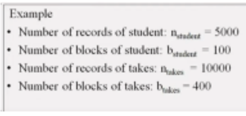
- The cost of the nested-loop join algorithm.
  - The number of pairs of tuples to be considered is $n_r * n_s$



Nested Loop Join

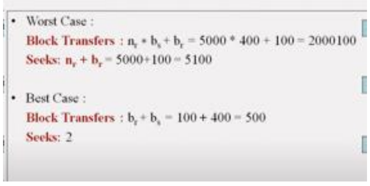
Worst case :

- The buffer can hold only one block of each relation
- A total block transfers = $n_r * b_s + b_r$
  where b_r and b_s denote the number of blocks containing tuples of r and s, respectively.
- Only one seek for each scan on the inner relation s since it is read sequentially, and a total of b_r seeks to read r
  Total seeks : $n_r + b_r$



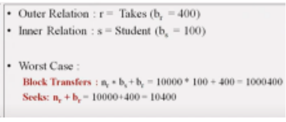
Example

- Number of records of student: $n_{student} = 5000$
- Number of blocks of student: $b_{student} = 100$
- Number of records of takes: $n_{takes} = 10000$
- Number of blocks of takes: $b_{takes} = 400$

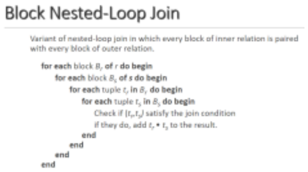


- Worst Case :
  Block Transfers : $n_r * b_s + b_r = 5000 * 400 + 100 = 2000100$
  Seeks: $n_r + b_r = 5000+100 = 5100$
- Best Case :
  Block Transfers : $b_r + b_s = 100 + 400 = 500$
  Seeks: 2

If we interchange the outer and inner block



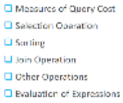
- Outer Relation : r = Takes (b_r = 400)
- Inner Relation : s = Student (b_s = 100)
- Worst Case :
  Block Transfers : $n_r * b_s + b_r = 10000 * 100 + 400 = 1000400$
  Seeks: $n_r + b_r = 10000+400 = 10400$



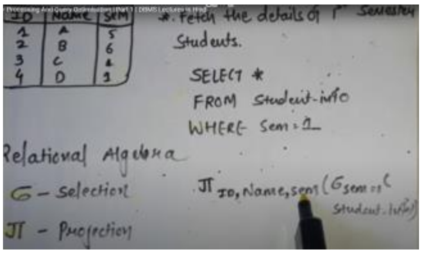
Block Nested-Loop Join

Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation.

```
for each block B_r of r do begin
    for each block B_s of s do begin
        for each tuple t_r in B_r do begin
            for each tuple t_s in B_s do begin
                Check if (t_r,t_s) satisfy the join condition
                if they do, add t_r • t_s to the result.
            end
        end
    end
end
```



- ❑ Measures of Query Cost
- ❑ Selection Operation
- ❑ Sorting
- ❑ Join Operation
- ❑ Other Operations
- ❑ Evaluation of Expressions



Processing And Query Optimization | Part 1 | DBMS Lectures in Hindi

ID	Name	Sem
1	A	5
2	B	6
3	C	4
4	D	1

*. Fetch the details of 1st Semester Students.

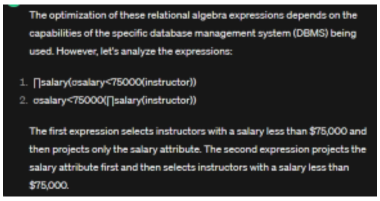
SELECT *
FROM Student-info
WHERE Sem = 1

Relational Algebra

σ – Selection

π – Projection

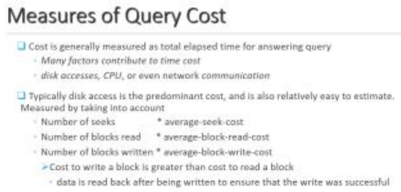
$\pi_{ID, Name, Sem}(\sigma_{Sem=1}(Student\_Info))$



The optimization of these relational algebra expressions depends on the capabilities of the specific database management system (DBMS) being used. However, let's analyze the expressions:

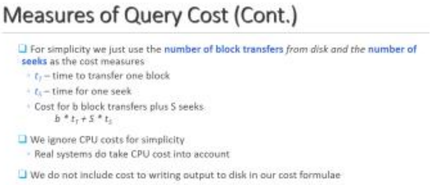
1. ∏salary(σsalary<75000(instructor))
2. σsalary<75000(∏salary(instructor))

The first expression selects instructors with a salary less than $75,000 and then projects only the salary attribute. The second expression projects the salary attribute first and then selects instructors with a salary less than $75,000.



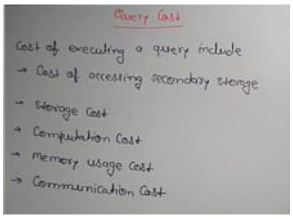
Measures of Query Cost

- ❑ Cost is generally measured as total elapsed time for answering query
  - *Many factors contribute to time cost*
  - *disk accesses, CPU, or even network communication*
- ❑ Typically disk access is the predominant cost, and is also relatively easy to estimate. Measured by taking into account
  - Number of seeks * average-seek-cost
  - Number of blocks read * average-block-read-cost
  - Number of blocks written * average-block-write-cost
    - ➢ Cost to write a block is greater than cost to read a block
      - data is read back after being written to ensure that the write was successful



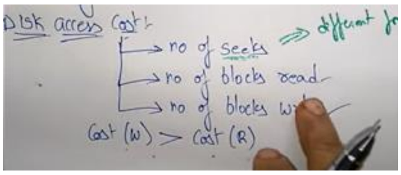
Measures of Query Cost (Cont.)

- ❑ For simplicity we just use the **number of block transfers** *from disk and the* **number of seeks** as the cost measures
  - t_T – time to transfer one block
  - t_S – time for one seek
  - Cost for b block transfers plus S seeks
    $b * t_T + S * t_S$
- ❑ We ignore CPU costs for simplicity
  - Real systems do take CPU cost into account
- ❑ We do not include cost to writing output to disk in our cost formulae

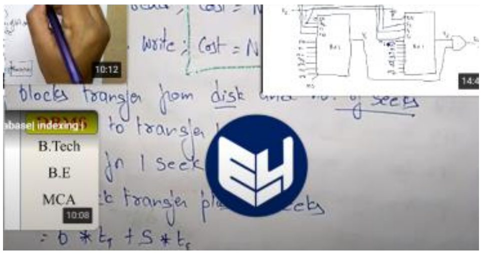


Query Cost

Cost of executing a query include
→ Cost of accessing secondary storage
→ Storage Cost
→ Computation Cost
→ memory usage Cost
→ Communication Cost

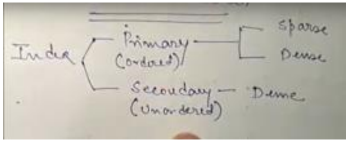


Disk access Cost:
→ no of Seeks ⇒ different fo
→ no of blocks read
→ no of blocks w
Cost (W) > Cost (R)

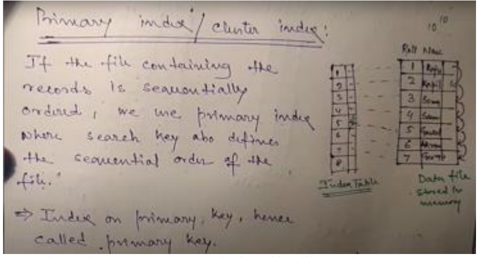


write : Cost = N
blocks transfer from disk and no. of seeks
to transfer
in 1 seek
transfer plus seeks
$= b * t_T + S * t_S$

Ordered Indices



- **Primary index**: in a sequentially ordered file, the index whose search key specifies the sequential order of the file.
- Also called **clustering index**
- The search key of a primary index is usually but not necessarily the primary key.



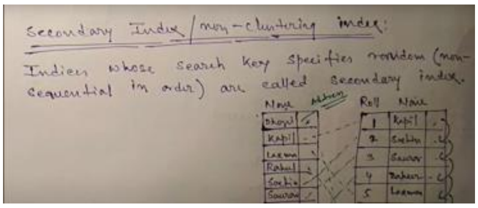
Primary index / cluster index:

"If the file containing the records is sequentially ordered, we use primary index where search key also defines the sequential order of the file."

⇒ Index on primary key, hence called primary key.

In Primary Index Attributes are in order means the data table will be in order

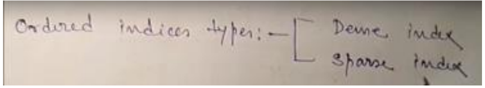
Secondary index: an index whose search key specifies an order different from the sequential order of the file. Also called **non-clustering index**



Secondary Index / non-clustering index:

Indices whose search key specifies random (non-sequential in order) are called secondary index.

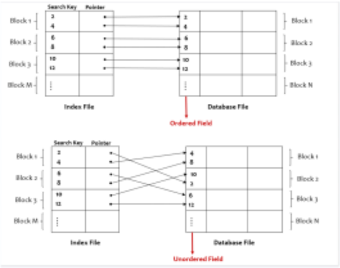
In Secondary index the attributes are not in order but the index table Is in order



Ordered indices types:— Dense index
Sparse index

In a dense index, each index entry points directly to the corresponding record in the data file. This allows for faster retrieval of specific records when searching for a particular key value. However, because every key is included in the index, it may require more storage space compared to a sparse index.

Dense Index can be built on order as well as unordered fields of the database files.



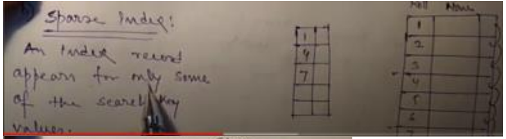
The terms "dense index" and "primary index" refer to different concepts in the context of database indexing:

1. **Dense Index:**
   - A dense index is a type of index where every possible key value has an entry in the index.
   - It contains pointers or references to every record in the data file, allowing for efficient retrieval of specific records based on key values.
   - Dense indexes are generally used to speed up retrieval operations, especially when searching for specific key values.

2. **Primary Index:**
   - A primary index, on the other hand, is a specific type of index that is created on the primary key of a table.
   - The primary key is a unique identifier for each record in the table, and the primary index is built on this key.
   - It ensures that the values in the primary key column are organized in a way that allows for fast retrieval of individual records.
   - Unlike a dense index, a primary index may or may not include every possible key value, depending on whether it is a clustered or non-clustered primary index.

In summary, the main difference lies in the focus and purpose: a dense index is a general concept referring to an index where every key has an entry, while a primary index specifically relates to the indexing structure built on the primary key of a table.

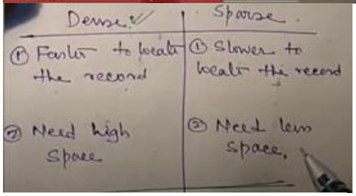
Sparse Index: contains index records for only some search-key values.



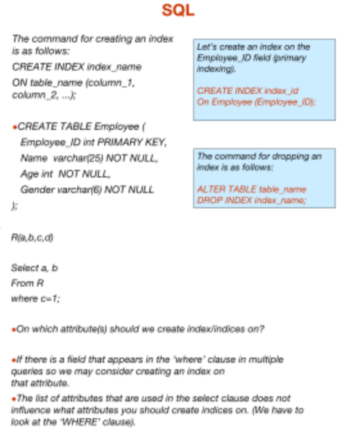
Sparse Index:

An index record appears for only some of the search key values.

1
4
7
Roll
Name
1
2
3
4
5
6



Dense ✓	Sparse
① Faster to locate the record	① Slower to locate the record
② Need high space	② Need less space,



SQL

The command for creating an index is as follows:

```
CREATE INDEX index_name
ON table_name (column_1,
column_2, ...);
```

Let's create an index on the Employee_ID field (primary indexing).

```
CREATE INDEX index_id
On Employee (Employee_ID);
```

```
•CREATE TABLE Employee (
  Employee_ID int PRIMARY KEY,
  Name  varchar(25) NOT NULL,
  Age int  NOT NULL,
  Gender varchar(6) NOT NULL
);
```

The command for dropping an index is as follows:

```
ALTER TABLE table_name
DROP INDEX index_name;
```

R(a,b,c,d)

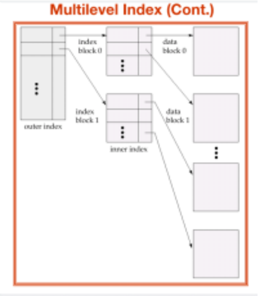
```
Select a, b
From R
where c=1;
```

- *On which attribute(s) should we create index/indices on?*

- *If there is a field that appears in the 'where' clause in multiple queries so we may consider creating an index on that attribute.*
- *The list of attributes that are used in the select clause does not influence what attributes you should create indices on. (We have to look at the 'WHERE' clause).*

Multilevel Index

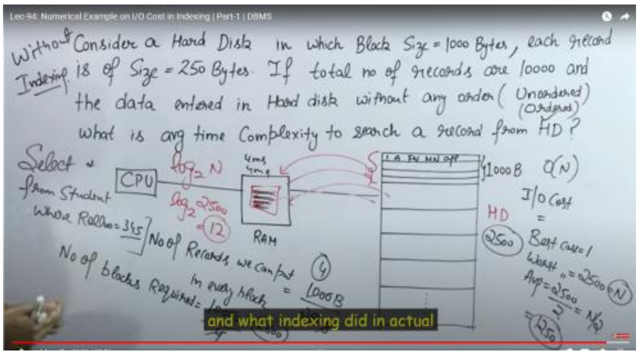
outer index – a sparse index of primary index
inner index – the primary index file



Multilevel Index (Cont.)

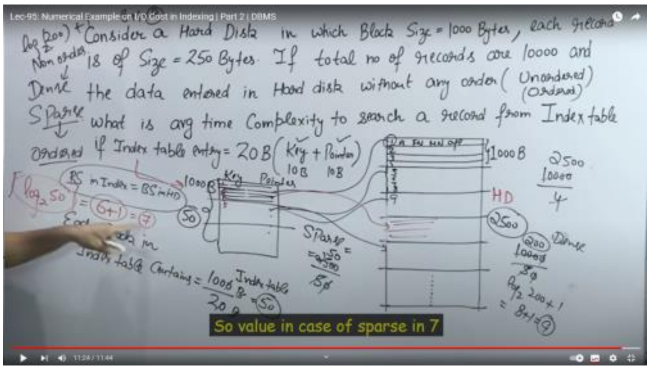
It can be said as **2-level Sparse Index**
We can 3-level ,4-level and so on..

Search Without Index wala topic reh gya hai



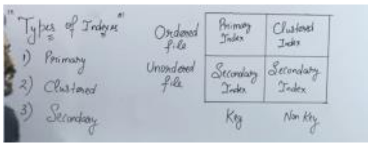
Lec-94: Numerical Example on I/O Cost in Indexing | Part-1 | DBMS

Without Indexing. Consider a Hard Disk in which Block Size = 1000 Bytes, each record is of Size = 250 Bytes. If total no of records are 10000 and the data entered in Hard disk without any order (Unordered) (Ordered) what is avg time Complexity to search a record from HD?



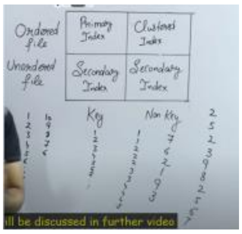
Lec-95: Numerical Example on I/O Cost in Indexing | Part 2 | DBMS

Consider a Hard Disk in which Block Size = 1000 Bytes, each record is of Size = 250 Bytes. If total no of records are 10000 and the data entered in Hard disk without any order (Unordered) (Ordered) what is avg time Complexity to search a record from Index table if Index table entry = 20 B (Key + Pointer)

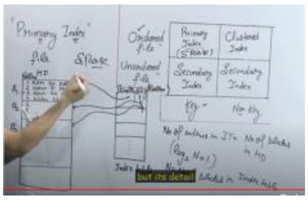


"Types of Indexes"
1) Primary
2) Clustered
3) Secondary

	Key	Non Key
Ordered file	Primary Index	Clustered Index
Unordered file	Secondary Index	Secondary Index



	Key	Non Key
Ordered file	Primary Index	Clustered Index
Unordered file	Secondary Index	Secondary Index



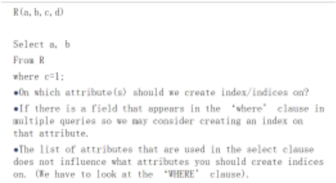
"Primary Index"

file Sparse

	Key	Non Key
Ordered file	Primary Index (Sparse)	Clustered Index
Unordered file	Secondary Index	Secondary Index

No of entries in IT = No of blocks in HD

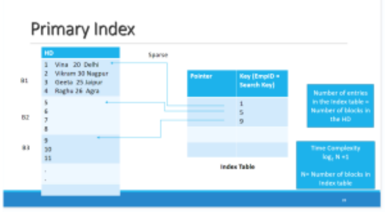
($\log_2 N + 1$)



R(a, b, c, d)

```
Select a, b
From R
where c=1;
```

- On which attribute(s) should we create index/indices on?
- If there is a field that appears in the 'where' clause in multiple queries so we may consider creating an index on that attribute.
- The list of attributes that are used in the select clause does not influence what attributes you should create indices on. (We have to look at the 'WHERE' clause).



Primary Index

HD
1 Vina 20 Delhi
2 Vikram 30 Nagpur
3 Geeta 25 Jaipur
4 Raghu 26 Agra
5
6
7
8
9
10
11
.
.

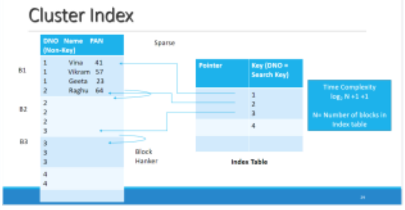
Pointer	Key (EmpID = Search Key)
	1
	5
	9

Index Table

Number of entries in the index table = Number of blocks in the HD

Time Complexity $\log_2 N + 1$

N= Number of blocks in Index table



Cluster Index

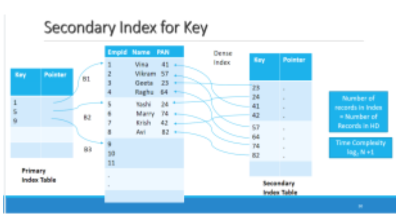
DNO (Non-Key)	Name	PAN
1	Vina	41
1	Vikram	57
1	Geeta	23
2	Raghu	64
2		
2		
2		
3		
3		
3		
3		
4		
4		

Pointer	Key (DNO = Search Key)
	1
	2
	3
	4

Index Table

Time Complexity $\log_2 N + 1 + 1$

N= Number of blocks in Index table



Secondary Index for Key

Key	Pointer
1	
5	
9	

Primary Index Table

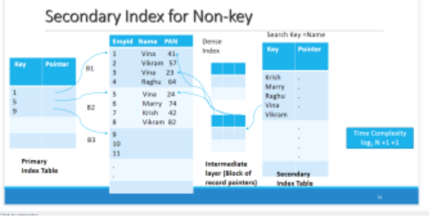
EmpId	Name	PAN
1	Vina	41
2	Vikram	57
3	Geeta	23
4	Raghu	64
5	Yashi	24
6	Marry	74
7	Krish	42
8	Avi	82
9		
10		
11		
.		
.		

Key	Pointer
23	.
24	.
41	.
42	.
57	.
64	.
74	.
82	.

Secondary Index Table

Number of records in Index = Number of Records in HD

Time Complexity $\log_2 N + 1$



Secondary Index for Non-key

Key	Pointer
1	
5	
9	

Primary Index Table

EmpId	Name	PAN
1	Vina	41
2	Vikram	57
3	Vina	23
4	Raghu	64
5	Vina	24
6	Marry	74
7	Krish	42
8	Vikram	82
9		
10		
11		
.		
.		

Search Key =Name

Key	Pointer
Krish	.
Marry	.
Raghu	.
Vina	.
Vikram	
	.
	.
	.
	.

Secondary Index Table

Intermediate layer (Block of record pointers)

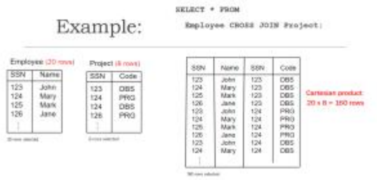
Time Complexity $\log_2 N + 1 + 1$

A JOIN is a means for combining fields from two tables by using values common to each (in most of the cases by using foreign key).

(SQL) Joins can be classified into the following categories :

- **Cartesian Products**
- **Inner Joins (Equijoins)**
- **Self Joins**
- **Outer Joins (Left, Right and Full)**

Cartesian Products



Example:

```
SELECT * FROM
  Employee CROSS JOIN Project;
```

Employee (20 rows)

SSN	Name
123	John
124	Mary
125	Mark
126	Jane

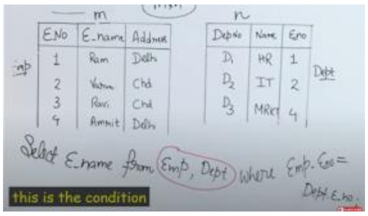
Project (8 rows)

SSN	Code
123	DBS
124	PRG
124	DBS
126	PRG

SSN	Name	SSN	Code
123	John	123	DBS
124	Mary	123	DBS
125	Mark	123	DBS
126	Jane	123	DBS
123	John	124	PRG
124	Mary	124	PRG
125	Mark	124	PRG
126	Jane	124	PRG
123	John	124	DBS
124	Mary	124	DBS

Cartesian product: 20 x 8 = 160 rows

Natural Join

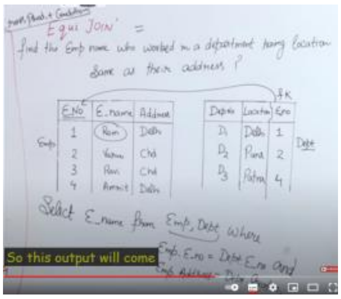


ENo	E_name	Address
1	Ram	Delhi
2	Varun	Chd
3	Ravi	Chd
4	Amrit	Delhi

Dept no	Name	Eno
D1	HR	1
D2	IT	2
D3	MRKT	4

Select E_name from Emp, Dept where Emp.Eno = Dept.E.no.

this is the condition



Equi JOIN =

find the Emp name who worked in a department having location same as their address ?

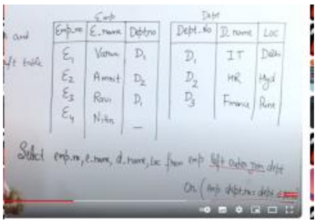
ENo	E_name	Address
1	Ram	Delhi
2	Varun	Chd
3	Ravi	Chd
4	Amrit	Delhi

Dept no	Location	Eno
D1	Delhi	1
D2	Pune	2
D3	Patna	4

Select E_name from Emp, Dept where Emp.E_no = Dept.E_no and Emp.Address = Dept.Lo...

So this output will come

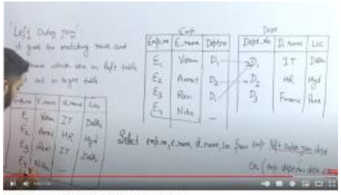
Left Outer Join



Emp_no	E_name	Dept no
E1	Varun	D1
E2	Amrit	D2
E3	Ravi	D1
E4	Nitin	—

Dept_No	D_name	Loc
D1	IT	Delhi
D2	HR	Hyd
D3	Finance	Pune

Select emp.no, e.name, d.name, loc from emp left outer join dept On (emp.deptno = dept.d...



'Left Outer Join'

it gives the matching rows and rows which are in left table but not in right table

Emp_no	E_name	Dept no
E1	Varun	D1
E2	Amrit	D2
E3	Ravi	D1
E4	Nitin	—

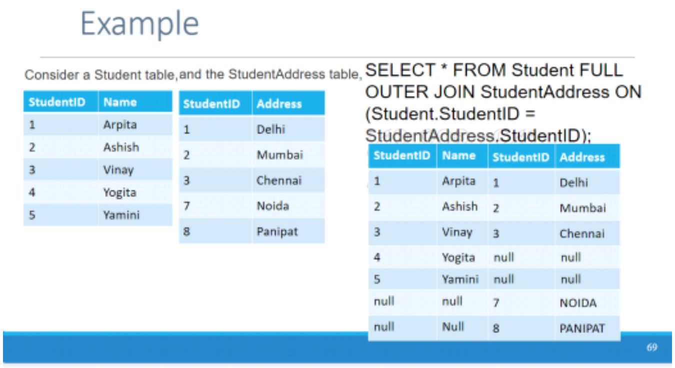
Dept_No	D_name	Loc
D1	IT	Delhi
D2	HR	Hyd
D3	Finance	Pune

Emp_no	E_name	d_name	Loc
E1	Varun	IT	Delhi
E2	Amrit	HR	Hyd
E3	Ravi	IT	Delhi
E4	Nitin		

Select emp.no, e.name, d.name, loc from emp left outer join dept On (emp.deptno = dept.d...

2: Left Outer Join operation with Example | Database Management System

Full Outer Join



Example

Consider a Student table, and the StudentAddress table,

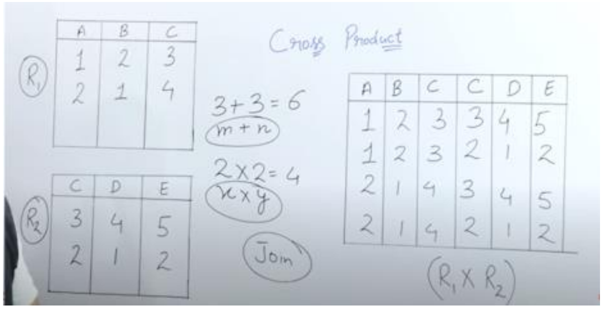
StudentID	Name
1	Arpita
2	Ashish
3	Vinay
4	Yogita
5	Yamini

StudentID	Address
1	Delhi
2	Mumbai
3	Chennai
7	Noida
8	Panipat

SELECT * FROM Student FULL OUTER JOIN StudentAddress ON (Student.StudentID = StudentAddress.StudentID);

StudentID	Name	StudentID	Address
1	Arpita	1	Delhi
2	Ashish	2	Mumbai
3	Vinay	3	Chennai
4	Yogita	null	null
5	Yamini	null	null
null	null	7	NOIDA
null	Null	8	PANIPAT

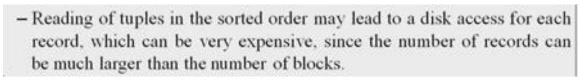
69



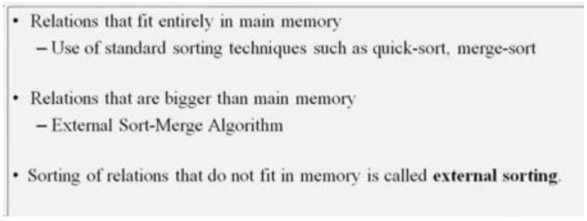
A	B	C
1	2	3
2	1	4

C	D	E
3	4	5
2	1	2

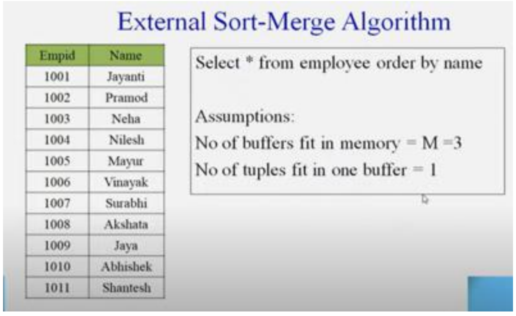
A	B	C	C	D	E
1	2	3	3	4	5
1	2	3	2	1	2
2	1	4	3	4	5
2	1	4	2	1	2



– Reading of tuples in the sorted order may lead to a disk access for each record, which can be very expensive, since the number of records can be much larger than the number of blocks.



- Relations that fit entirely in main memory
  – Use of standard sorting techniques such as quick-sort, merge-sort
- Relations that are bigger than main memory
  – External Sort-Merge Algorithm
- Sorting of relations that do not fit in memory is called **external sorting**.



External Sort-Merge Algorithm

Empid	Name
1001	Jayanti
1002	Pramod
1003	Neha
1004	Nilesh
1005	Mayur
1006	Vinayak
1007	Surabhi
1008	Akshata
1009	Jaya
1010	Abhishek
1011	Shantesh

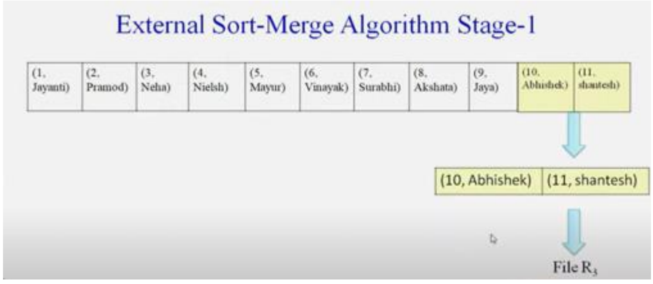
Select * from employee order by name

Assumptions:

No of buffers fit in memory = M =3

No of tuples fit in one buffer = 1

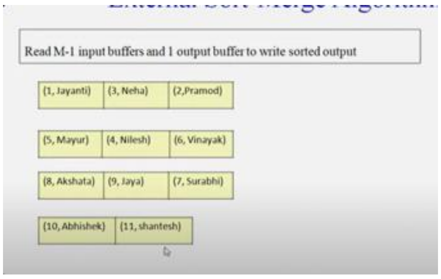
Sorting according to the name we take 3-3 batch and sort them according to the name



External Sort-Merge Algorithm Stage-1

(1, Jayanti)	(2, Pramod)	(3, Neha)	(4, Nielsh)	(5, Mayur)	(6, Vinayak)	(7, Surabhi)	(8, Akshata)	(9, Jaya)	(10, Abhishek)	(11, shantesh)

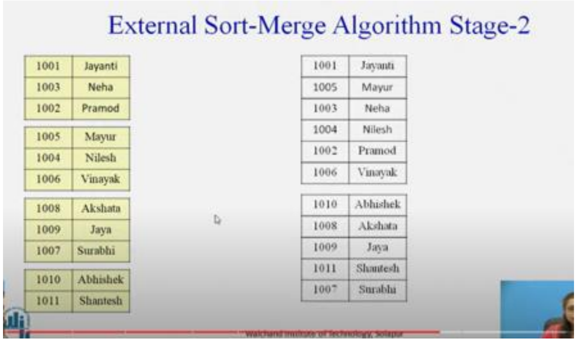
(10, Abhishek)	(11, shantesh)



External Sort-Merge Algorithm

Read M-1 input buffers and 1 output buffer to write sorted output

(1, Jayanti)	(3, Neha)	(2,Pramod)
(5, Mayur)	(4, Nilesh)	(6, Vinayak)
(8, Akshata)	(9, Jaya)	(7, Surabhi)
(10, Abhishek)	(11, shantesh)	



External Sort-Merge Algorithm Stage-2

1001	Jayanti
1003	Neha
1002	Pramod

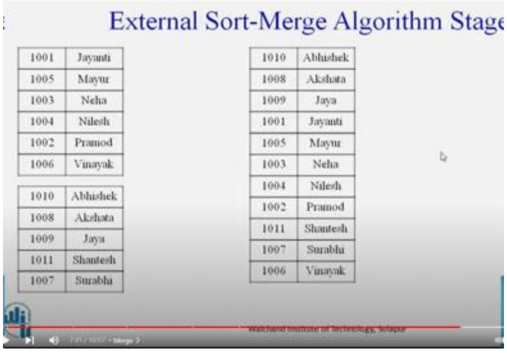
1005	Mayur
1004	Nilesh
1006	Vinayak

1008	Akshata
1009	Jaya
1007	Surabhi

1010	Abhishek
1011	Shantesh

1001	Jayanti
1005	Mayur
1003	Neha
1004	Nilesh
1002	Pramod
1006	Vinayak

1010	Abhishek
1008	Akshata
1009	Jaya
1011	Shantesh
1007	Surabhi

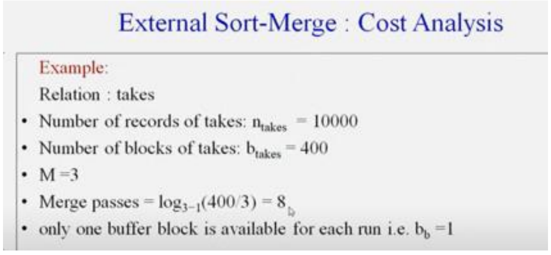


External Sort-Merge Algorithm Stage

1001	Jayanti
1005	Mayur
1003	Neha
1004	Nilesh
1002	Pramod
1006	Vinayak

1010	Abhishek
1008	Akshata
1009	Jaya
1011	Shantesh
1007	Surabhi

1010	Abhishek
1008	Akshata
1009	Jaya
1001	Jayanti
1005	Mayur
1003	Neha
1004	Nilesh
1002	Pramod
1011	Shantesh
1007	Surabhi
1006	Vinayak

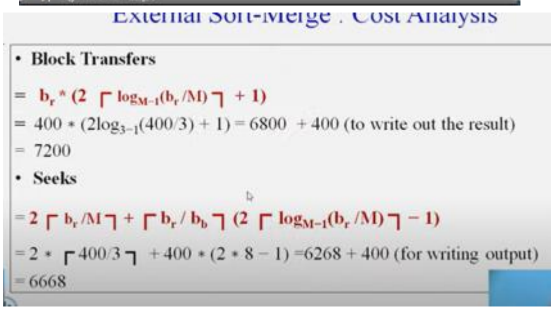


External Sort-Merge : Cost Analysis

Example:

Relation : takes

- Number of records of takes: n_{takes} = 10000
- Number of blocks of takes: b_{takes} = 400
- M =3
- Merge passes = $\log_{3-1}(400/3) = 8$
- only one buffer block is available for each run i.e. b_b =1



External Sort-Merge : Cost Analysis

- **Block Transfers**

$= b_r * (2 \lceil \log_{M-1}(b_r/M) \rceil + 1)$

$= 400 * (2\log_{3-1}(400/3) + 1) = 6800 + 400$ (to write out the result)

$= 7200$

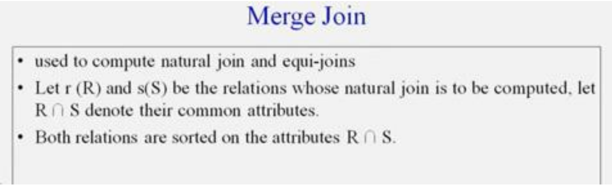
- **Seeks**

$= 2 \lceil b_r/M \rceil + \lceil b_r / b_b \rceil (2 \lceil \log_{M-1}(b_r/M) \rceil - 1)$

$= 2 * \lceil 400/3 \rceil + 400 * (2 * 8 - 1) = 6268 + 400$ (for writing output)

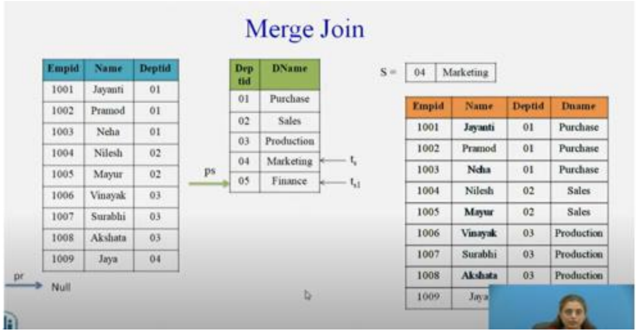
$= 6668$

MERGE JOIN



Merge Join

- used to compute natural join and equi-joins
- Let r (R) and s(S) be the relations whose natural join is to be computed, let R ∩ S denote their common attributes.
- Both relations are sorted on the attributes R ∩ S.



Merge Join

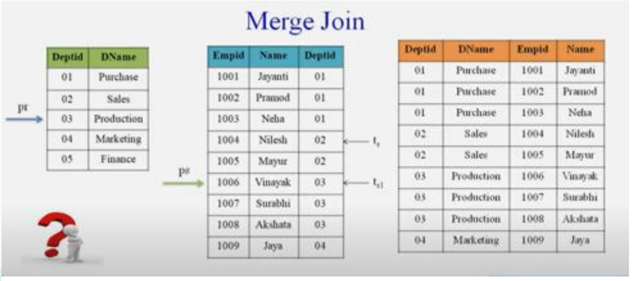
Empid	Name	Deptid
1001	Jayanti	01
1002	Pramod	01
1003	Neha	01
1004	Nilesh	02
1005	Mayur	02
1006	Vinayak	03
1007	Surabhi	03
1008	Akshata	03
1009	Jaya	04

Deptid	DName
01	Purchase
02	Sales
03	Production
04	Marketing
05	Finance

S =

04	Marketing

Empid	Name	Deptid	Dname
1001	Jayanti	01	Purchase
1002	Pramod	01	Purchase
1003	Neha	01	Purchase
1004	Nilesh	02	Sales
1005	Mayur	02	Sales
1006	Vinayak	03	Production
1007	Surabhi	03	Production
1008	Akshata	03	Production
1009	Jaya		

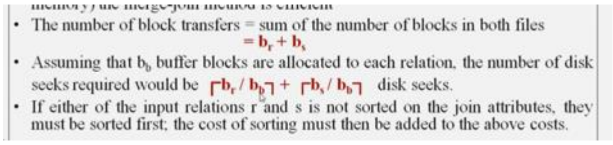


Merge Join

Deptid	DName
01	Purchase
02	Sales
03	Production
04	Marketing
05	Finance

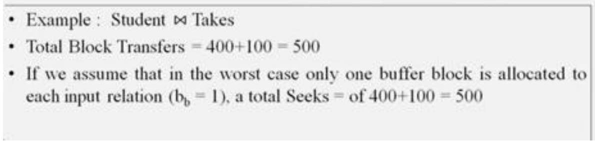
Empid	Name	Deptid
1001	Jayanti	01
1002	Pramod	01
1003	Neha	01
1004	Nilesh	02
1005	Mayur	02
1006	Vinayak	03
1007	Surabhi	03
1008	Akshata	03
1009	Jaya	04

Deptid	DName	Empid	Name
01	Purchase	1001	Jayanti
01	Purchase	1002	Pramod
01	Purchase	1003	Neha
02	Sales	1004	Nilesh
02	Sales	1005	Mayur
03	Production	1006	Vinayak
03	Production	1007	Surabhi
03	Production	1008	Akshata
04	Marketing	1009	Jaya



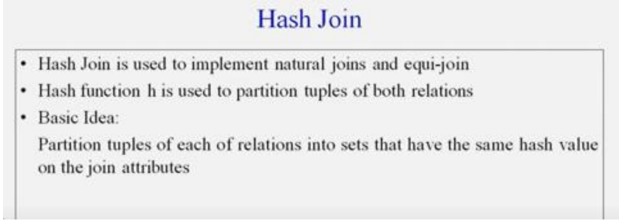
memory) the merge-join method is efficient

- The number of block transfers = sum of the number of blocks in both files $= b_r + b_s$
- Assuming that b_b buffer blocks are allocated to each relation, the number of disk seeks required would be $\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$ disk seeks.
- If either of the input relations r and s is not sorted on the join attributes, they must be sorted first; the cost of sorting must then be added to the above costs.



- Example : Student ⋈ Takes
- Total Block Transfers = 400+100 = 500
- If we assume that in the worst case only one buffer block is allocated to each input relation ($b_b = 1$), a total Seeks = of 400+100 = 500

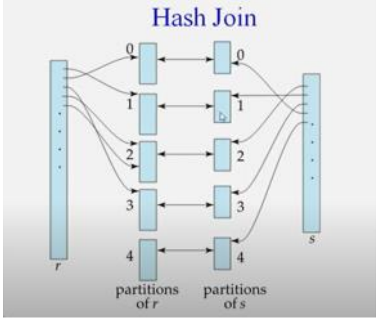
Merge join is much fast as campared to the other join provided that the table will be sorted



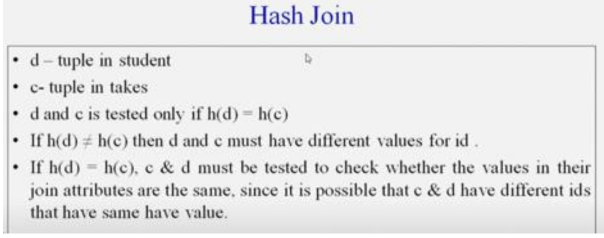
Hash Join

- Hash Join is used to implement natural joins and equi-join
- Hash function h is used to partition tuples of both relations
- Basic Idea:

  Partition tuples of each of relations into sets that have the same hash value on the join attributes

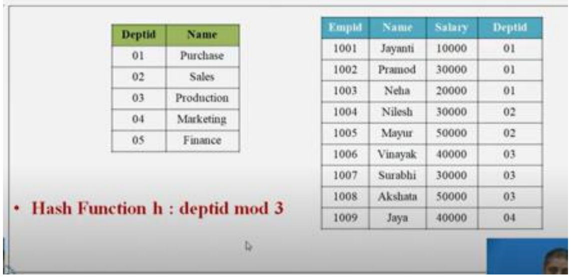


Hash Join



Hash Join

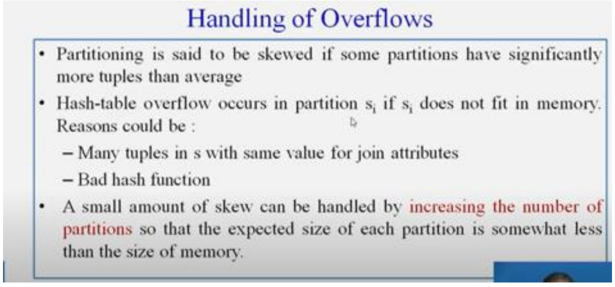
- d – tuple in student
- c- tuple in takes
- d and c is tested only if h(d) = h(c)
- If h(d) ≠ h(c) then d and c must have different values for id .
- If h(d) = h(c), c & d must be tested to check whether the values in their join attributes are the same, since it is possible that c & d have different ids that have same have value.



Deptid	Name
01	Purchase
02	Sales
03	Production
04	Marketing
05	Finance

Empid	Name	Salary	Deptid
1001	Jayanti	10000	01
1002	Pramod	30000	01
1003	Neha	20000	01
1004	Nilesh	30000	02
1005	Mayur	50000	02
1006	Vinayak	40000	03
1007	Surabhi	30000	03
1008	Akshata	50000	03
1009	Jaya	40000	04

- **Hash Function h : deptid mod 3**



Handling of Overflows

- Partitioning is said to be skewed if some partitions have significantly more tuples than average
- Hash-table overflow occurs in partition s_i if s_i does not fit in memory. Reasons could be :
  - Many tuples in s with same value for join attributes
  - Bad hash function
- A small amount of skew can be handled by increasing the number of partitions so that the expected size of each partition is somewhat less than the size of memory.