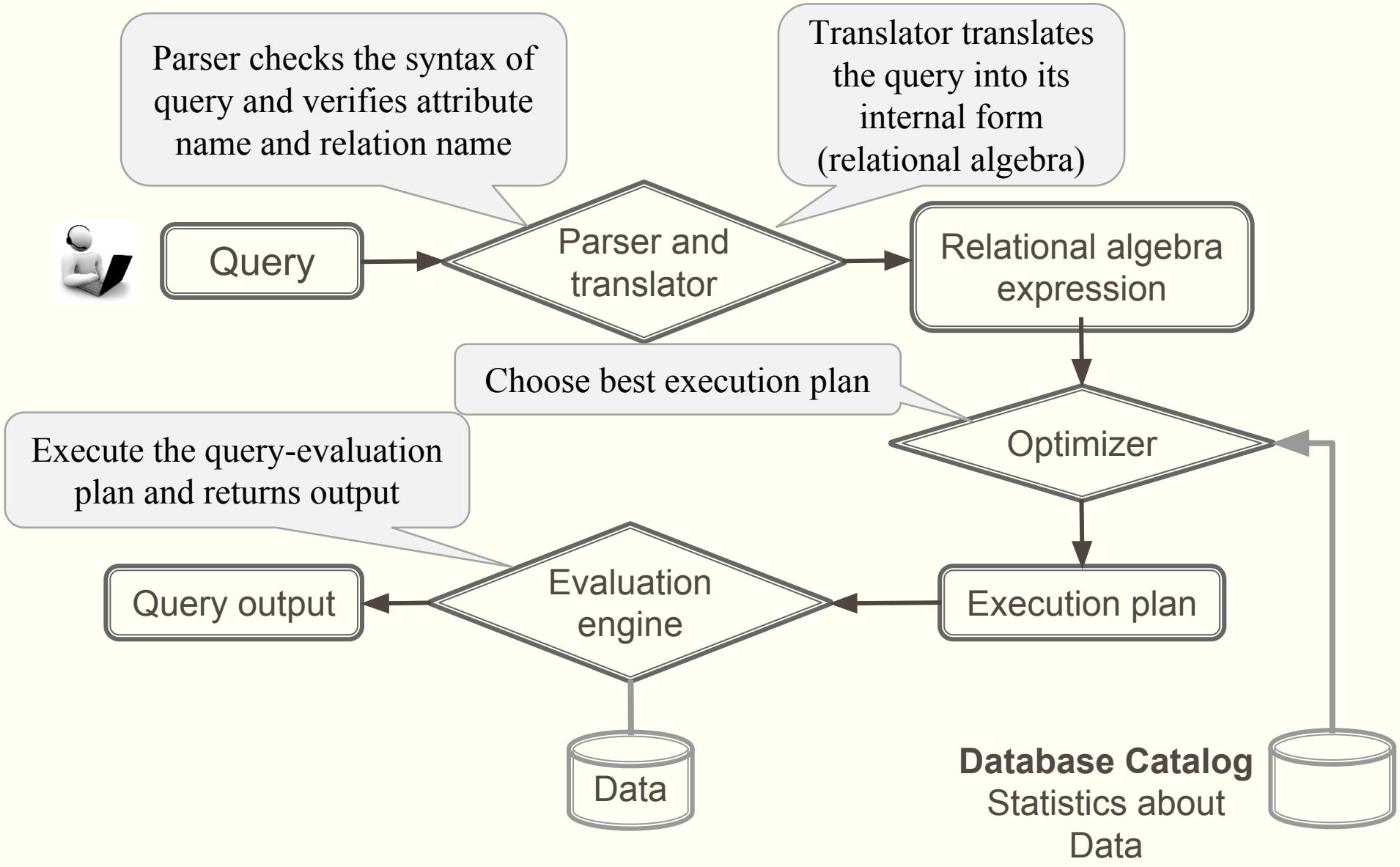
QUERY PROCESSING AND OPTIMIZATION

Unit-5

Steps in Query Processing



Measures of Query Cost

- Cost is generally measured as **the total time required to execute a statement/query**.
- Factors contribute to time cost
  - **Disk accesses** (time to process a data request and retrieve the required data from the storage device)
  - **CPU time to execute a query**
  - **Network communication cost**
- **Disk access is the predominant (major) cost, since disk access is slow as compared to in-memory operation**.
- **Cost to write a block is greater than cost to read a block** because data is read back after being written to ensure that the write was successful.

Selection Operator

- Symbol: σ (Sigma)
- Notation: $\sigma_{condition}$ (Relation)
- Operation: **Selects tuples** from a relation that **satisfy a given condition**.
- Operators: =, <>, <, >, <=, >=, Λ (AND), V (OR)

Example Branch belongs to "CE"

Student

RollNo	Name	Branch	SPI
101	Raju	CE	8
102	Mitesh	ME	9
103	Nilesh	CI	9
104	Meet	CE	9

Answer

Output

RollNo	Name	Branch	SPI
101	Raju	CE	8
104	Meet	CE	9

Search algorithm for selection operation

- Linear search (A1)
- Binary search (A2)

Linear search (A1)

- It **scans each blocks** and **tests all records** to see whether they **satisfy the selection condition**.
  - Cost of linear search (worst case) = **br**

    br denotes number of blocks containing records from relation r

- If the **selection condition is there on a (primary) key attribute**, then **system can stop searching if the required record is found**.
  - cost of linear search (best case) = (br /2)

- If the **selection is on non (primary) key** attribute then **multiple block may contains required records**, then **the cost of scanning such blocks need to be added to the cost estimate**.

- Linear search can be applied regardless of
  - **selection condition** or
  - **ordering of records** in the file (relation)

- This algorithm is **slower than binary search** algorithm.

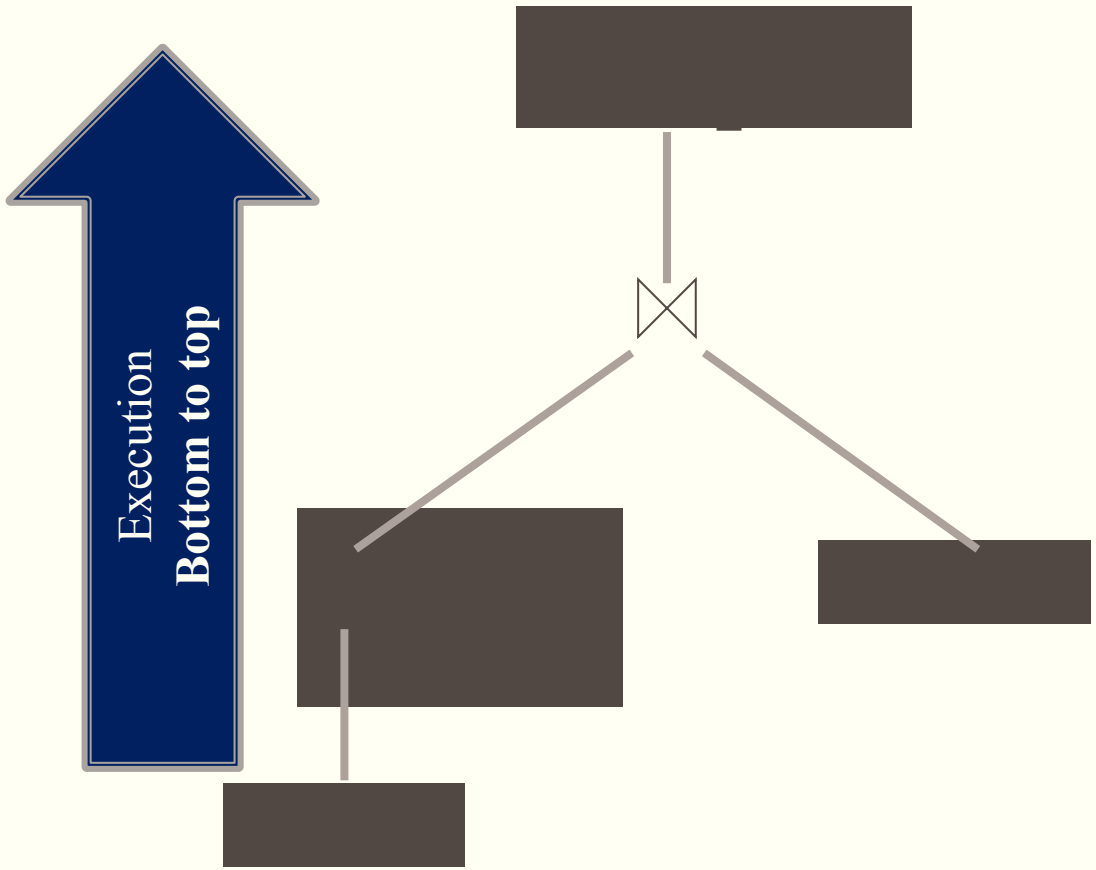
Binary search (A2)

- Generally, this algorithm is used if **selection is an equality comparison on the (primary) key attribute and file (relation) is ordered (sorted) on (primary) key attribute**.
- cost of binary search = **[log2(br)]**
  - br denotes number of blocks containing records from relation r
- This algorithm is **faster than linear search** algorithm.

Evaluation of expressions

- Expression may contain more than one operations, solving expression will be difficult if it contains more than one operations.

- To evaluate such expression we need to **evaluate each operations one by one** in appropriate order.
- Two **methods for evaluating an expression carrying multiple operations are**:
  - Materialization
  - Pipelining



Materialization

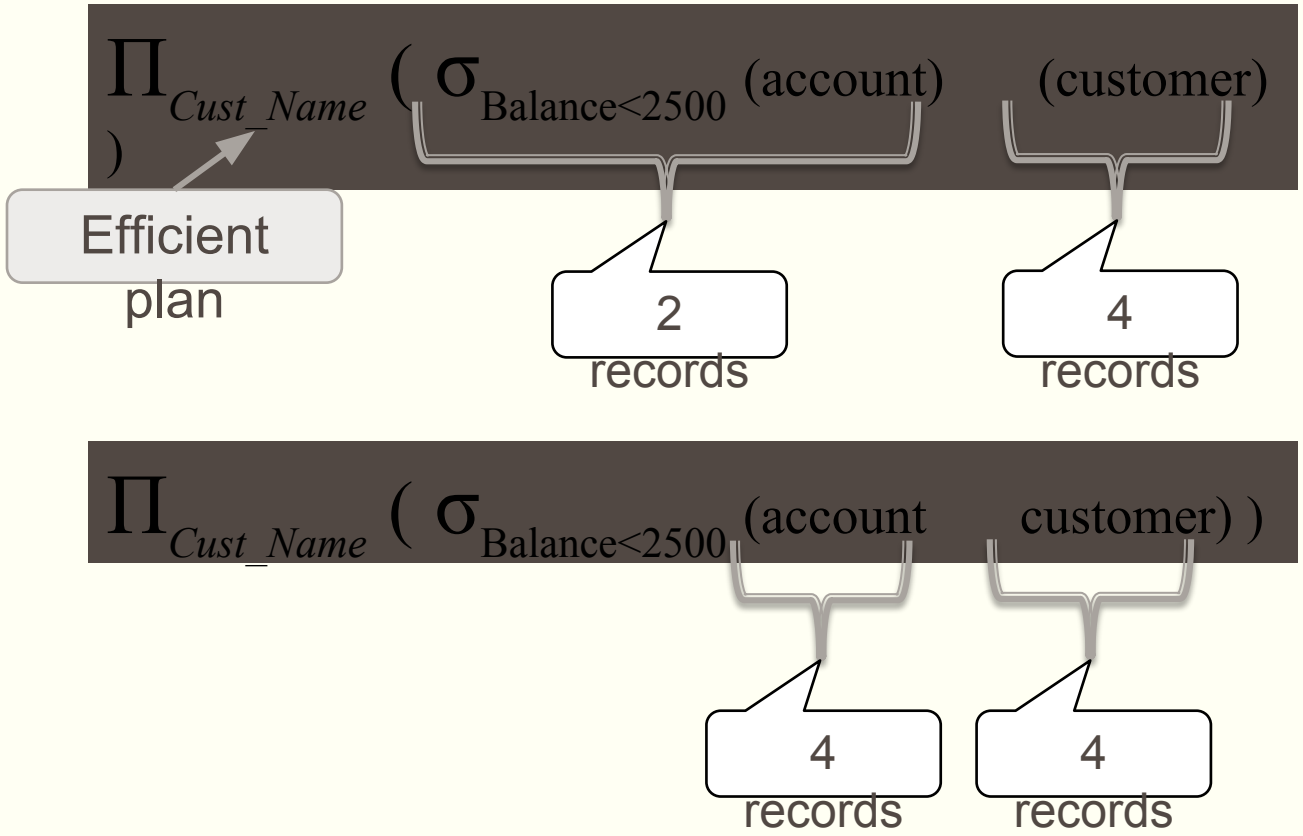
- Materialization **evaluates the expression tree** of the relational algebra operation **from the bottom** and **performs the innermost or leaf-level operations first**.
- The **intermediate result of each operation is materialized (store in temporary relation)** and **becomes input for subsequent (next) operations**.
- The **cost of materialization** is the **sum of the individual operations plus the cost of writing the intermediate results to disk**.
- The problem with materialization is that
  - it **creates lots of temporary relations**
  - it **performs lots of I/O operations**

Pipelining

- In pipelining, **operations form a queue, and results are passed from one operation to another** as they are calculated.
- **To reduce number of intermediate temporary relations, we pass results of one operation to the next operation in the pipelines**.
- Combining operations into a pipeline **eliminates the cost of reading and writing temporary relations**.
- Pipelines can be executed in two ways:
  - **Demand driven** (System makes repeated requests for tuples from the operation at the top of pipeline)
  - **Producer driven** (Operations do not wait for request to produce tuples, but generate the tuples eagerly.)

Query optimization

- It is a **process of selecting the most efficient query evaluation plan from the available possible plans**.



Customer

CID	ANO	Name
C01	A01	Raj
C02	A02	Meet
C03	A03	Jay
C04	A04	Ram

Account

ANO	Balance
A01	3000
A02	1000
A03	2000
A04	4000

Approaches to Query Optimization

- Exhaustive Search Optimization
  - **Generates all possible query plans** and then the **best plan is selected**.
  - It **provides best solution**.
- Heuristic Based Optimization
  - Heuristic **based optimization uses rule-based optimization approaches** for query optimization.
  - **Performs select and project operations before join operations**. This is done by moving the select and project operations down the query tree. This reduces the number of tuples available for join.
  - **Avoid cross-product operation** because they result in very large-sized intermediate tables.
  - This **algorithms do not necessarily produce the best query plan**.

Transformation of relational expressions

- Two relational algebra expressions are said to be **equivalent if the two expressions generate the same set of tuples**.
- Example:

Customer

CID	ANO	Name
C01	A01	Raj
C02	A02	Meet
C03	A03	Jay
C04	A04	Ram

Account

ANO	Balance
A01	3000
A02	1000
A03	2000
A04	4000

$$\Pi_{Name}\left(\sigma_{Balance<2500}(Account) \bowtie (Customer)\right) = \Pi_{Name}\left(\sigma_{Balance<2500}(Account \bowtie Customer)\right)$$

Customer

Name
Meet
Jay

Transformation of relational expressions

- **Combined selection operation can be divided** into sequence of individual selections. This transformation is called **cascade of σ**.
- Example:

Customer

CID	ANO	Name	Balance
C01	1	Raj	3000
C02	2	Meet	1000
C03	3	Jay	2000
C04	4	Ram	4000

$\sigma_{ANO<3 \Lambda Balance<2000}$ (Customer)

=

$\sigma_{ANO<3}$ ($\sigma_{Balance<2000}$ (Customer))

OUTPUT

Output

CID	ANO	Name	Balance
C02	2	Meet	1000

$$\sigma_{\theta 1 \Lambda \theta 2}(E) = \sigma_{\theta 1}(\sigma_{\theta 2}(E))$$

Transformation of relational expressions

- **Selection operations are commutative.**
- Example:

Customer

CID	ANO	Name	Balance
C01	1	Raj	3000
C02	2	Meet	1000
C03	3	Jay	2000
C04	4	Ram	4000

$\sigma_{ANO<3}(\sigma_{Balance<2000}(Customer))$

=

$\sigma_{Balance<2000}(\sigma_{ANO<3}(Customer))$

OUTPUT

Output

CID	ANO	Name	Balance
C02	2	Meet	1000

$$\sigma_{\theta 1}(\sigma_{\theta 2}(E)) = \sigma_{\theta 2}(\sigma_{\theta 1}(E))$$

Transformation of relational expressions

- If **more than one projection operation** is used in expression then **only the outer projection operation is required**. So **skip** all the other **inner projection operation**.
- Example:

Customer

CID	ANO	Name	Balance
C01	1	Raj	3000
C02	2	Meet	1000
C03	3	Jay	2000
C04	4	Ram	4000

$\Pi_{Name}(\Pi_{ANO, Name}(Customer))$

=

$\Pi_{Name}(Customer)$

OUTPUT

Customer

Name
Raj
Meet
Jay
Ram

$$\Pi_{L1}(\Pi_{L2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L1}(E)$$

Transformation of relational expressions

- **Selection operation can be joined with Cartesian product and theta join.**
- Example:

Customer

CID	ANO	Name
C01	1	Raj
C02	2	Meet
C03	3	Jay
C04	4	Ram

Account

ANO	Balance
1	3000
2	1000
3	2000
4	4000

$\sigma_{ANO<3}$ (Customer ⋈ Account)

=

(Customer) ⋈ $\sigma_{ANO<3}$ (Account)

OUTPUT

Output

CID	ANO	Name	Balance
C01	1	Raj	3000
C02	2	Meet	1000

$$\sigma_{\theta}(E1 \bowtie E2)) = E1 \bowtie_{\theta} E2$$

$$\sigma_{\theta 1}(E1 \bowtie_{\theta 2} E2)) = E1 \bowtie_{\theta 1 \wedge \theta 2} E2$$

Transformation of relational expressions

- **Theta operations are commutative.**
- Example:

Customer

CID	ANO	Name
C01	1	Raj
C02	2	Meet
C03	3	Jay
C04	4	Ram

Account

ANO	Balance
1	3000
2	1000
3	2000
4	4000

$(Account) \bowtie \sigma_{ANO<3} (Customer)$

=

$(Customer) \bowtie \sigma_{ANO<3} (Account)$

OUTPUT

Output

CID	ANO	Name	Balance
C01	1	Raj	3000
C02	2	Meet	1000

$$E1 \bowtie \sigma_{\theta} E2 = E2 \bowtie \sigma_{\theta} E1$$

Transformation of relational expressions

- **Natural join operations are associative.**

$$(E1 \bowtie E2) \bowtie E3 \quad = \quad E1 \bowtie (E2 \bowtie E3)$$

- **Selection operation distribute over theta join operation** under the following condition
  - When all the attributes in the selection condition θ0 involves only the attributes of the one of the expression (says E1) being joined.

$$\sigma_{\theta 0} (E1 \bowtie_{\theta} E2) \quad = \quad (\sigma_{\theta 0} (E1)) \bowtie_{\theta} E2$$

  - When the selection condition θ1 involves only the attributes of E1 and θ2 involves only the attributes of E2.

$$\sigma_{\theta 1 \wedge \theta 2} (E1 \bowtie_{\theta} E2) \quad = \quad (\sigma_{\theta 1}(E1) \bowtie_{\theta} (\sigma_{\theta 2} (E2)))$$

Transformation of relational expressions

- Set operations union and intersection are commutative.

Customer
Cst_Name
Raj
Meet

Employee
Emp_Name
Meet
Suresh

Customer U Employee

=

Employee U Customer

OUTPUT

Output
Name
Raj
Meet
Suresh

Customer
Cst_Name
Raj
Meet

Employee
Emp_Name
Meet
Suresh

Customer ∩ Employee

=

Employee ∩ Customer

OUTPUT

Output
Name
Meet

E1 U E2 = E2 U E1 & E1 ∩ E2 = E2 ∩ E1

Set difference is not commutative

Transformation of relational expressions

- Set operations **union and intersection are associative.**

Customer
Cst_Name
Raj
Meet

Employee
Emp_Name
Meet
Suresh

Student
Stu_Name
Raj
Meet

(Customer U Employee) U Student

=

Customer U (Employee U Student)

OUTPUT

Output
Name
Raj
Meet
Suresh

Customer
Cst_Name
Raj
Meet

Employee
Emp_Name
Meet
Suresh

Student
Stu_Name
Raj
Meet

(Customer ∩ Employee) ∩ Student

=

Customer ∩ (Employee ∩ Student)

OUTPUT

Output
Name
Meet

(E1 U E2) U E3 = E1 U (E2 U E3) & (E1 ∩ E2) ∩ E3 = E1 ∩ (E2 ∩ E3)

Transformation of relational expressions

- **Selection operation distributes over U, ∩ and –.**

$$\sigma_{\theta}(E1 \cup E2) = \sigma_{\theta}(E1) \cup \sigma_{\theta}(E2)$$

$$\sigma_{\theta}(E1 \cap E2) = \sigma_{\theta}(E1) \cap \sigma_{\theta}(E2)$$

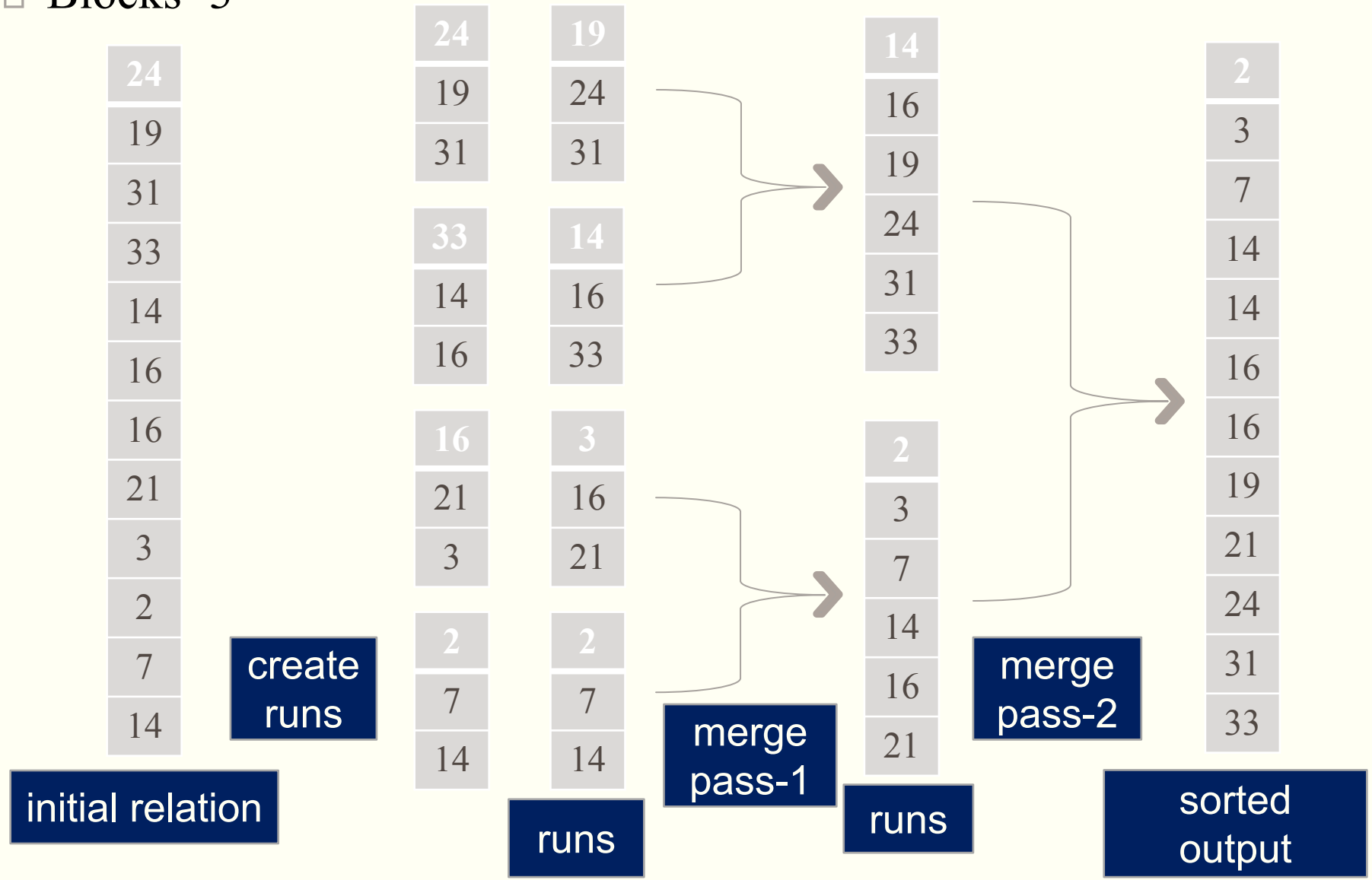
$$\sigma_{\theta}(E1 - E2) = \sigma_{\theta}(E1) - \sigma_{\theta}(E2)$$

Sorting

- Several of the relational operations, such as **joins, can be implemented efficiently if the input relations are first sorted**.
- We can **sort a relation by building an index on the relation** and then using that index to read the relation in sorted order.
- Such a process orders the relation only logically rather than physically.
- Hence reading of tuples in the sorted order may lead to disk access for each record, which can be very expensive.
- So it is desirable to order the records physically.
- **Sorting of relation that fit into main memory**, standard sorting techniques such as **quick-sort can be used**.
- **Sorting of relations that do not fit in main memory is called external sorting**.
- Most commonly used **algorithm** for this type of sorting is **external sort merge algorithm**.

External Sort-Merge (Example)

- Blocks=3



External Sort-Merge (Algorithm)

- Let M denote memory size (in pages).

1. Create sorted runs. Let i be 0 initially.
   - Repeatedly do the following till the end of the relation:
     1) Read M blocks of relation into memory
     2) Sort the in-memory blocks
     3) Write sorted data to run Ri; then increment i.
   - Let the final value of i be N
2. Merge the runs (N-way merge). We assume (for now) that N < M.
   - Use N blocks of memory to buffer input runs, and 1 block to buffer output. Read the first block of each run into its buffer page
   - repeat

     Select the first record (in sort order) among all buffer pages

     Write the record to the output buffer. If the output buffer is full write it to disk.

     Delete the record from its input buffer page.
     - If the buffer page becomes empty then read the next block (if any) of the run into the buffer.
   - until all input buffer pages are empty:

External Sort-Merge (Algorithm)

- If $N \geq M$, several merge passes are required.
  - In each pass, contiguous groups of M - 1 runs are merged.
  - A pass reduces the number of runs by a factor of M -1, and creates runs longer by the same factor.
    - E.g. If M=11, and there are 90 runs, one pass reduces the number of runs to 9, each 10 times the size of the initial runs
  - Repeated passes are performed till all runs have been merged into one.

Sum (Nested loop join)

- Assuming **worst case** memory availability and the following given statistics for the relations customer and depositor
  - Number of records of customer: 10,000 ($n_{customer}$)
  - Number of records of depositor: 5,000 ($n_{depositor}$)
  - Number of blocks of customer: 400 ($b_{customer}$)
  - Number of blocks of depositor: 100 ($b_{depositor}$)
- Estimate the cost
  1. with depositor as outer relation
  2. with customer as outer relation

Sum (Nested loop join)

- Assuming **worst case** memory availability and the following given statistics for the relations customer and depositor
  - Number of records of customer: 10,000 ($n_{customer}$)
  - Number of records of depositor: 5,000 ($n_{depositor}$)
  - Number of blocks of customer: 400 ($b_{customer}$)
  - Number of blocks of depositor: 100 ($b_{depositor}$)
- Estimate the cost
  1. with depositor as outer relation

     No. of blocks access = $n_{depositor} * b_{customer} + b_{depositor}$

     $= 5000 * 400 + 100$

     $= 2000100$

Sum (Nested loop join)

- Assuming **worst case** memory availability and the following given statistics for the relations customer and depositor
  - Number of records of customer: 10,000 ($n_{customer}$)
  - Number of records of depositor: 5,000 ($n_{depositor}$)
  - Number of blocks of customer: 400 ($b_{customer}$)
  - Number of blocks of depositor: 100 ($b_{depositor}$)
- Estimate the cost
  1. with customer as outer relation

     No. of blocks access = $n_{customer} * b_{depositor} + b_{customer}$

     = 10000 * 100 + 400

     = 1000400

Sum (Nested loop join)

- Assuming **best case** memory availability and the following given statistics for the relations customer and depositor
  - Number of records of customer: 10,000 ($n_{customer}$)
  - Number of records of depositor: 5,000 ($n_{depositor}$)
  - Number of blocks of customer: 400 ($b_{customer}$)
  - Number of blocks of depositor: 100 ($b_{depositor}$)
- Estimate the cost
  1. with customer as outer relation

     No. of blocks access = $b_{depositor} + b_{customer}$

     $= 100 + 400$

     $= 500$

Join Operations

- There are several different algorithms that can be used to implement joins
  - Nested-Loop Join
  - Block Nested-Loop Join
  - Index Nested-Loop Join
  - Sort-Merge Join
  - Hash-Join

Cost of computing for all joins

- **R is the outer** and **S is the inner relation** of the join.
  - Number of records of R: (N_R)
  - Number of records of S: (N_S)
  - Number of blocks of R: (B_R)
  - Number of blocks of S: (B_S)

Join	Worst Case	Best Case
Nested-Loop Join	$B_R + N_R * B_S$	$B_R + B_S$
Block Nested-Loop Join	$B_R + B_R * B_S$	$B_R + B_S$
Index Nested-Loop Join	$B_R + N_R * c$	
Merge Join	$B_R + B_S$	
Hash-Join	$3 * (B_R + B_S)$	

  - c is the cost of a single selection on S using the join condition.

Questions asked in GTU

1. Explain query processing steps. **OR** Discuss various steps of query processing with proper diagram.
2. Explain Heuristics in optimization.
3. Explain the purpose of sorting with example with reference to query optimization.
4. Explain the measures of finding out the cost of a query in query processing.

Thank you…!!