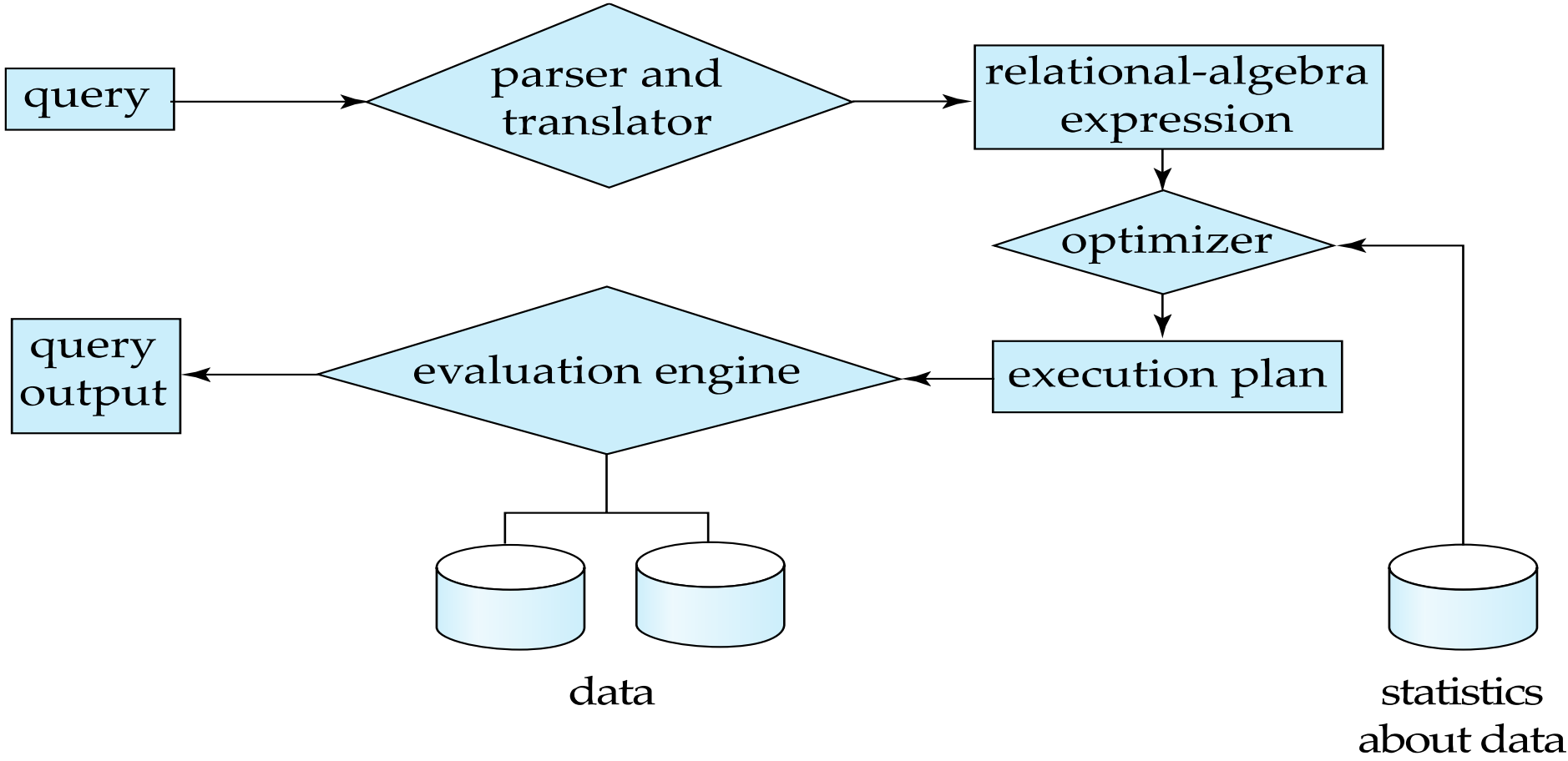
Query Processing

Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Basic Steps in Query Processing (Cont.)

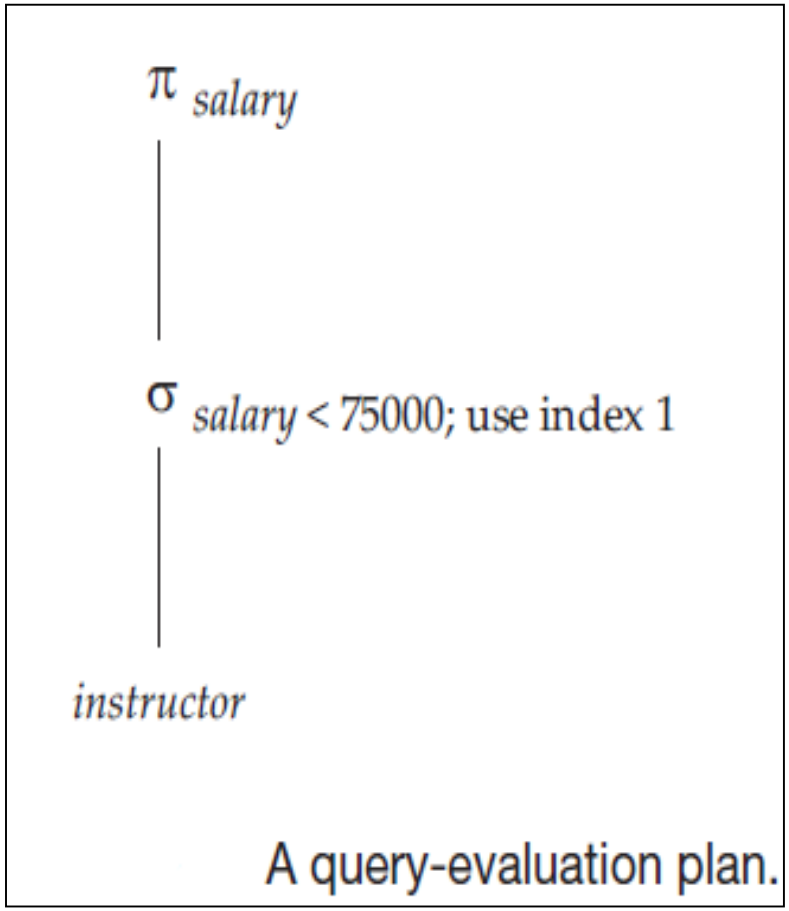
- Parsing and translation
  - Parser checks syntax, verifies relations.
  - The system constructs a parse-tree representation of the query,
  - translate the query into its internal form. This is then translated into relational algebra.
  - If the query was expressed in terms of a view, the translation phase also **replaces all uses of the view** by the relational-algebra expression.
- **Query Optimization**
  - The different evaluation plans for a given query can have different costs. It is the responsibility of the system to construct a query evaluation plan that **minimizes the cost of query evaluation**; this task is called **query optimization**.
- **Evaluation**
  - The query-execution engine takes a query-evaluation plan, **executes that plan**, and returns the answers to the query.

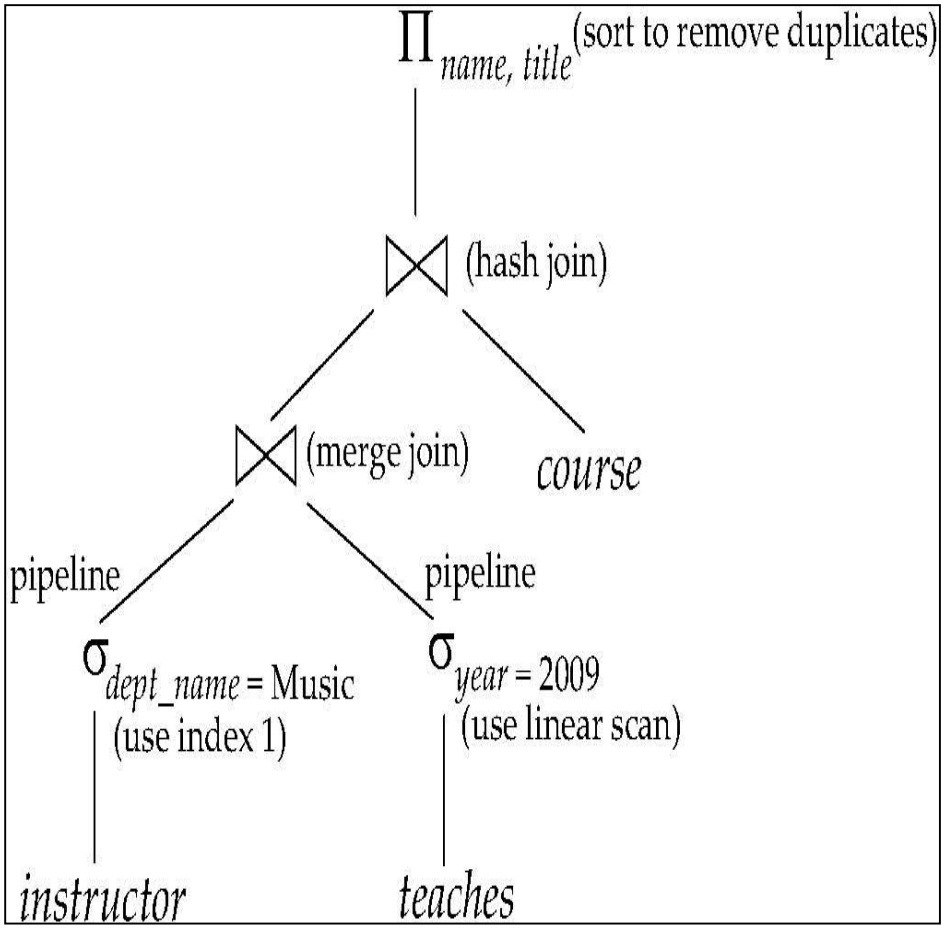
Basic Steps in Query Processing : Optimization

- A relational algebra expression may have many equivalent expressions
  - E.g., $\sigma_{salary<75000}(\Pi_{salary}(instructor))$ is equivalent to $\Pi_{salary}(\sigma_{salary<75000}(instructor))$
- Each relational algebra operation can be evaluated using one of several different algorithms
  - Correspondingly, a relational-algebra expression can be evaluated in many ways.
- Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**.
  - Annotations may state the algorithm to be used for a specific operation, or the particular index or indices to use.
- An annotated instructions in a relational algebra is called an **evaluation primitive**.
- A **sequence of primitive operations** that can be used to evaluate a query is a **query-execution plan** or **query-evaluation plan**.

Evaluation Plan

E.g., We can use an index on *salary* to find instructors with salary < 75000, **or** can perform complete relation scan and discard instructors with salary ≥ 75000





Annotated expression specifying detailed evaluation strategy is called an evaluation-plan

Basic Steps: Optimization (Cont.)

- **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.

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
  - We want to transfer **b** blocks of data and assume we require **S** seeks to transfer **b** blocks of data..
  - 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

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 = b_r block transfers $*t_T + 1* t_s$

  - b_r denotes number of blocks containing records from relation *r*
  - If selection is on a key attribute, can stop on finding record

    cost = (b_r /2) block transfers $*t_T + 1* t_s$

    - (Because -at most 1 record satisfies the search criteria)

Join Operation

- Several different algorithms to implement joins
  - Nested-loop join
  - Block nested-loop join
  - Indexed nested-loop join
  - Merge-join
  - Hash-join
- Choice based on cost estimate
- Examples use the following information ***student*** $\bowtie$ ***takes***
  - Number of **records** of *student*: n_r= 5,000

    takes: n_s= 10,000
  - Number of **blocks** of *student*: b_r= 100

    takes: b_s=400

Student

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

takes

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-101	1	Fall	2009	C
12345	CS-190	2	Spring	2009	A
12345	CS-315	1	Spring	2010	A
12345	CS-347	1	Fall	2009	A
19991	HIS-351	1	Spring	2010	B
23121	FIN-201	1	Spring	2010	C+
44553	PHY-101	1	Fall	2009	B-
45678	CS-101	1	Fall	2009	F
45678	CS-101	1	Spring	2010	B+
45678	CS-319	1	Spring	2010	B
54321	CS-101	1	Fall	2009	A-
54321	CS-190	2	Spring	2009	B+
55739	MU-199	1	Spring	2010	A-
76543	CS-101	1	Fall	2009	A
76543	CS-319	2	Spring	2010	A
76653	EE-181	1	Spring	2009	C
98765	CS-101	1	Fall	2009	C-
98765	CS-315	1	Spring	2010	B
98988	BIO-101	1	Summer	2009	A
98988	BIO-301	1	Summer	2010	null

Nested-Loop Join

- To compute the theta join $r \bowtie_\theta s$

  *For each record in **r**, we have to perform complete scan on s.*
  pairs of tuples to be considered is $n_r * n_s$, where n_r denotes the number of tuples in r, and n_s denotes the number of tuples in 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 θ
      if they do, **add** $t_r \bullet t_s$ to the result.
    end
  end

- r is called the **outer relation** and s the **inner relation** of the join.
- Requires no indices and can be used with any kind of join condition.
- Expensive since it examines every pair of tuples in the two relations.

Nested-Loop Join (Cont.)

- In the worst case, if there is enough memory only to hold one block of each relation, the estimated cost is
  - $n_r * b_s + b_r$ block transfers, and
  - $n_r + b_r$ seeks*

records of *student*: n_r= 5,000
takes: n_s= 10,000
blocks of *student*: b_r= 100
takes: b_s=400

- Example: Assuming **worst case** memory availability cost estimate is
  - with **student as outer relation**: (student has 100 blocks, smaller relation than takes, hence costlier)
    - 5000 * 400 + 100 **= 2,000,100** block transfers,
    - 5000 + 100 **= 5100** seeks,

Nested-Loop Join (Cont.)

records of *student*: n_r= 5,000
takes: n_s= 10,000
blocks of *student*: b_r= 100
takes: b_s=400

- with ***takes*** **as the outer relation** (takes has 400 blocks, bigger relation than student, The number of block transfers is significantly less, and although the number of seeks is higher, the overall cost is reduced, assuming seek time $\mathbf{t_S}$ **= 4 milliseconds** and transfer time $\mathbf{t_T}$ **= 0.1 milliseconds.** hence costlier)
  - 10000 ∗ 100 + 400 = **1,000,400** block transfers and **10,400** seeks

Nested-Loop Join (Cont.)

- If the **any one relation fits entirely in memory**, use that as the **inner relation.**
  - Reduces cost to **b_r + b_s block transfers** and **2 seeks**
- If **smaller relation (*student*) fits entirely in memory**, the cost estimate will be 500 block transfers.

*** See notes section.**

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 $t_r \bullet t_s$ to the result.
      end
    end
  end
end

Block Nested-Loop Join (Cont.)

- **Worst case** estimate: $b_r * b_s + b_r$ block transfers and $2 * b_r$ seeks
  - Each block in the inner relation *s* is read once for each *block* in the outer relation.

- **Best case**: If the smaller relation fits entirely in memory
  - $b_r + b_s$ block transfers & **2** seeks.

Example of Block Nested-Loop Join Costs

- Compute ***student*** ⋈ ***takes***, with *student* as the outer relation.
  - Number of records of *student*: $n_{student}$ =**n_r= 5, 000**.
  - Number of blocks of *student*: $b_{student}$ = **b_r=100.**
  - Number of records of *takes*: n_{takes} =**n_s= 10, 000**.
  - Number of blocks of *takes*: b_{takes} = **b_s =400.**
- Cost of block nested loops join

  *cost =b_r * b_s + b_r* block transfers and 2 * b_r seeks
  - 400*100 + 100 = 40,100 block transfers and
  - 2 * 100 = 200 seeks
    - assuming worst case memory
    - may be significantly less with more memory

End of Chapter