Query Processing and Optimization

TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

Week - 3 ,Topic – 2, Lesson 5 and 6

Query Processing and Optimization

Outline

EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

- Query optimization techniques (rule-based + practical)
- Cost-based optimization and execution plans
- Use case with examples

Pipelining (Streaming)

TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

- **Iterator model**: each operator produces the next tuple on demand (next()).
- **Pipelining**: pass tuples directly from one operator to the next (no full intermediate table).
  - Pipelines can be executed in two ways: demand driven and producer driven
- **Materialization**: write intermediate results to disk/temp table (slower, but sometimes required).
- Pipeline breaks: sort, hash build phase, or blocking aggregates often need full input.

```
Example pipeline:
Scan → Filter (σ) → Project (π) → Output
```

SELECT Name FROM Student WHERE Marks > 80;

Implementation of demand-driven pipelining

Each operation is implemented as an **iterator** implementing the following operations

- **open()**
  - file scan: initialize file scan
  - state: pointer to beginning of file
  - merge join: sort relations;
  - state: pointers to beginning of sorted relations
- **next()**
  - for file scan: Output next tuple, and advance and store file pointer
  - for merge join: continue with merge from earlier state till next output tuple is found. Save pointers as iterator state.
- **close()**

3: What is Query Optimization?

3. Query optimization techniques

EIU
TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

- Optimization chooses a good plan among many equivalent plans.
- Logical optimization: rewrite query without changing meaning (algebraic rules).
- Physical optimization: pick access paths + join algorithms + join order.
- Two styles: rule-based (heuristics) and cost-based (use a cost model).

Classic Heuristic Rules

Simple rules that often help

TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

- Push selections down: apply σ as early as possible (reduce rows early).
- Push projections down: keep only needed columns (reduce row width).
- Combine selections: $\sigma_{c1}(\sigma_{c2}(R)) \rightarrow \sigma_{c2 \wedge c2}(R)$
- Reorder joins (when safe): join smaller / filtered relations first.
- Replace Cartesian Products: Use joins with conditions instead of cross products.
- Query Tree: a graphical representation of the operators, relations, attributes and predicates and processing sequence during query processing. It is composed of three main parts:
  - The Leafs: the base relations used for processing the query/ extracting the required information.
  - The Root: the final result/relation as an output based on the operation on the relations used for query processing
  - Nodes: intermediate results or relations before reaching the final result.
  - Sequence of execution of operation in a query tree will start from the leaves and continues to the intermediate nodes and ends at the root.

Example Cont…

Example-

Before optimization

SELECT E.name FROM Employee E, Department D
WHERE E.dept_id = D.dept_id AND D.location = 'Binh Doung';

Problem

- Cartesian product produces huge intermediate results
- Selection is applied too late

π name
|
⋈ E.dept_id = D.dept_id
/ \
Employee σ location=' Binh Doung '
|
Department

After Optimization

SELECT E.name FROM Employee E JOIN Department D ON
E.dept_id = D.dept_id WHERE D.location = ' Binh Doung ';

Query tree –

π name
|
σ E.dept_id = D.dept_id AND D.location=' Binh Doung '
|
Employee × Department

Another Example (Push Selection)

Same result, usually faster

TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

```
Original (conceptual):
R ⋈ S  then  σ_{R.x > 10}

Rewrite:
(σ_{R.x > 10}(R)) ⋈ S
```

- Filtering R first shrinks the join input.
- Less data enters the join → fewer comparisons and less I/O.

Physical Aids (Indexes & Views)

Optimization uses what exists in storage

EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

- Indexes can turn scans into fast lookups (B+ tree, hash index).
- Clustered index: table stored in index order (good for ranges).
- Materialized view: precomputed result stored on disk.
- Partitioning: split table to scan only relevant parts.

Optimizer chooses plans based on available indexes/views + estimated benefit.

Statistics & Selectivity (Basics)

Estimates guide the optimizer

TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

- DBMS stores stats: #rows, #pages, distinct values, histograms.
- Selectivity estimates how many rows pass a predicate.
- Common assumptions: uniform distribution and independence (not always true).
- Better stats → better plan choices (especially for join order).

Example: if attribute A has V distinct values, sel(A = c) ≈ 1 / V (uniform).

4: Cost-Based Optimization (CBO)

4. Cost-based optimization and execution plans

- CBO picks the plan with the lowest estimated cost.
- Cost is usually dominated by disk I/O, then CPU, Network cost (in distributed databases), then memory.
- DBMS uses table/index statistics to estimate: cardinalities and costs.
- The chosen plan is an execution plan (physical operator tree).
- Highly efficient and Adapts to data size and distribution

Techniques Used

- Evaluates multiple query plans
- Uses statistics about tables and indexes

Chooses optimal:

- ❑ Join order
- ❑ Join method (Nested Loop, Hash Join, Merge Join)
- ❑ Access path (Index scan vs. Table scan)

A Simple Cost Model

Not exact—just useful for comparisons

- I/O Cost : Represents the cost of reading and writing data pages from disk. Since disk access is slow, I/O cost is usually the largest contributor to total cost.
  - ➢ I/O cost = Number of page reads / writes × Cost per I/O operation
- CPU Cost: Represents the processing cost of evaluating tuples (rows). Includes comparisons, joins, filtering, and aggregation.
  - ➢ CPU cost = Number of tuples processed × Cost per tuple operation
- Memory Cost (Impact): Memory is not always counted directly.

If sufficient memory is available:

- ✓ Operations like sorting and hashing stay in memory → **lower I/O cost**

If memory is insufficient:

- ✓ Intermediate results are written to disk → **higher I/O cost**

The total cost of an execution plan is often estimated using:

$$Cost(plan) = (N_{IO} \times t_{IO}) + (N_{CPU} \times t_{CPU})$$

Where

N_{IO} = Number of page I/O operations

t_{IO} = Time per I/O operation

N_{CPU} = Number of tuple (row) operations

t_{CPU} = Time per CPU operation

Cost Based Optimization Example

TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

Assume
Employee has 1,000 rows
Department has 10 rows
Index exists on Employee.dept_id

Query -
SELECT name, dept_name FROM Employee E, Department D WHERE E.dept_id = D.dept_id;

Possible Execution Plans
Plan 1: Nested Loop Join (Employee → Department)
- Outer table: Employee (1,000 rows)
- Inner table: Department (10 rows)
- Cost ≈ 1,000 × 10 = 10,000 comparisons

Plan 2: Index Join (Department → Employee)
- Outer table: Department (10 rows)
- Use index on Employee.dept_id
- Cost ≈ 10 index lookups = very low

Optimizer chooses Plan 2 because it has lower estimated cost based on:
- Table size
- Index availability
- Number of comparisons

⋈ (Index Join)
/ \
Department Employee (using index)

Name	Dept_name
Kim	IT
Nygen	IT
Tien	HR

More Example….

TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

Assume the database system uses the following costs:
Time per I/O operation
t_{IO} = 5 ms
Time per CPU operation
t_{CPU} = 0.01

Plan A: Full Table Scan
Number of page I/Os
N_{IO} =200
Number of tuple operations
N_{CPU} = 10000

Sol: Cost(A) = (200×5) + (10,000×0.01)
= 1,000 + 100
= 1,100 ms

Plan B: Index Scan
Number of page I/Os
N_{IO} = 50
Number of tuple operations
N_{CPU} = 2,000

Sol:
Cost(B) = (50×5) + (2,000×0.01)
= 250+20
=270 ms

emp_id	name
101	Kim
105	Nyugen

Searching Join Orders (Idea Only)

Enumerating plans without exploding

EIU
TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

- Join order matters a lot (A ⋈ B) ⋈ C may be much faster than A ⋈ (B ⋈ C).
- Optimizers often search “left-deep” trees (one join at a time).
- Dynamic programming idea: reuse best plan for each subset of tables.
- Heuristics/pruning stop the search from becoming too large.

Key point: the optimizer explores alternatives and keeps the cheapest (estimated) ones.

Reading an Execution Plan

What you see in EXPLAIN output

TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

- Operators: Seq Scan, Index Scan, Hash Join, Sort, Aggregate, etc.
- Estimated rows/cardinality at each node.
- Estimated cost (often “startup..total”).
- Actual runtime (if you run EXPLAIN ANALYZE) may differ from estimates.

```
Mini example (conceptual):
Hash Join
  -> Seq Scan on R
  -> Hash
       -> Index Scan on S
```

Employee Use Case Example -

CREATE TABLE Employee (emp_id INT, name VARCHAR(20), dept_id INT, salary INT);

CREATE TABLE Department (dept_id INT, dept_name VARCHAR(20), location VARCHAR(20));

INSERT INTO Employee VALUES (1, ‘Kim', 10, 60000), (2, ‘Nyugen', 20, 45000), (3, ‘Nahi', 10, 70000), (4, ‘Tien', 30, 40000);

INSERT INTO Department VALUES (10, 'IT', ‘Binh Dung’), (20, 'HR', ‘Ho Chi Minh City’), (30, 'Sales', ‘Hanoi');

Classical JOIN Query:
SELECT E.name, D.dept_name FROM Employee E, Department D WHERE E.dept_id = D.dept_id AND D.location = ' Binh Dung ‘ AND E.salary > 50000;

Heuristic (Rule-Based) Optimization - Optimized SQL (Logical)

Rule Applied - Combine selections, Push selection down, Replace Cartesian product with JOIN and Project required columns early

SELECT E.name, D.dept_name FROM Employee E JOIN Department D ON E.dept_id = D.dept_id WHERE D.location = ' Binh Dung ’ AND E.salary > 50000;

Employee Use Case Example -

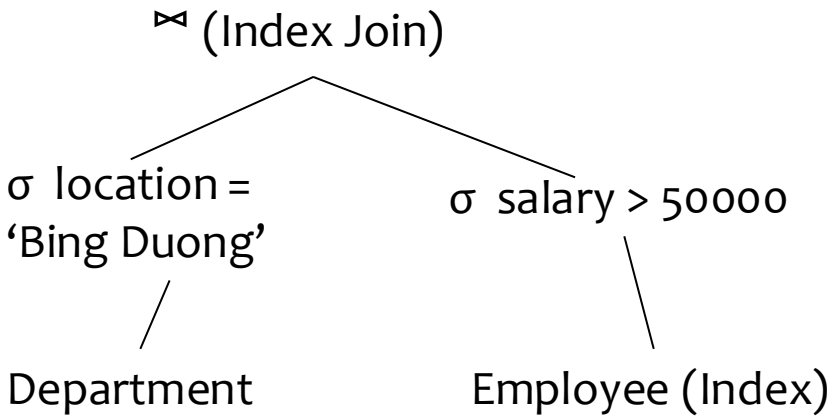
TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

Cost-Based Optimization:
Assume information-

- Employee table = 1000 rows (200 pages)
- Department table = 50 rows (5 pages)
- Index on Employee.dept_id
- Disk I/O cost t_{IO} = 5 ms
- CPU cost t_{CPU} = 0.01 ms

Plan B: Index Join (Chosen Plan)
N_{IO} = 55 pages
N_{CPU} = 2,000 tuple operations
Cost(B) = (55×5)+(2,000×0.01)
=275+20
=295 ms
Cost-based optimizer selects Plan B

Final Query Execution plan



Plan A: Full Table Scan + Nested Loop Join
Solutions-
N_{IO} = 205 pages
N_{CPU} = 10,000 tuple operations

Cost(A) = (205×5) + (10,000×0.01)
= 1025+100
= 1125 ms

Revision

TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

- Query processing: parse → rewrite → optimize → execute.
- Joins have multiple algorithms; pick based on size, indexes, and predicates.
- Optimization uses rules + statistics to reduce work early.
- Cost-based optimizers compare many plans and choose the cheapest estimate.

Tip for practice: write a query, guess the plan, then check with EXPLAIN in a DBMS.

Further Reading and Practice

EIU
TRƯỜNG ĐẠI HỌC QUỐC TẾ MIỀN ĐÔNG
EASTERN INTERNATIONAL UNIVERSITY

* Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). Database System Concepts (7th ed.). McGraw-Hill. (Part six, Chapter 16. Query optimization page no-743 to 794)
* Rupley Jr, Michael L. "Introduction to query processing and optimization." Indiana University at South Bend (2008).

* Practice Relational database with SQL compiler Such as MySQL, PostgreSQL, SQL Workbench
  * Online compiler - https://sqlfiddle.com/

    https://sqlzoo.net/wiki/SQL_Tutorial