Query processing

Learning Maps

Sequence	Title
1	Introduction to databases
2	Relational Databases
3	Relational Algebra
4	Structured Query Language – Part 1
5	Structured Query Language – Part 2
6	Constraints and Triggers
7	Entity Relationship Model
8	Functional Dependency
9	Normalization
10	Storage - Indexing
11	Query Processing
12	Transaction Management – Part 1
13	Transaction Management – Part 2

Intro > Overview

☐ A : Voice and PPT Overview
☐ B : Text-based Overview
■ C : Video and PPT Overview

Opening Message	→ In this lesson, we will study →Query processing →Query optimization
Lesson topic	1. Course overview 2. Basic concepts on database 3. Data management: database vs. file approach
Learning Goals	Upon completion of this lesson, students will be able to: 1. Recall the concepts of database, DBMS, data model, file system. 2. Identify the characteristics of database and file system approach in data management

Intro > Keywords

Keyword	Description
Query processing	Activities involved in retrieving/storing data from/to the database
Query optimization	Selection of an efficient query execution plan

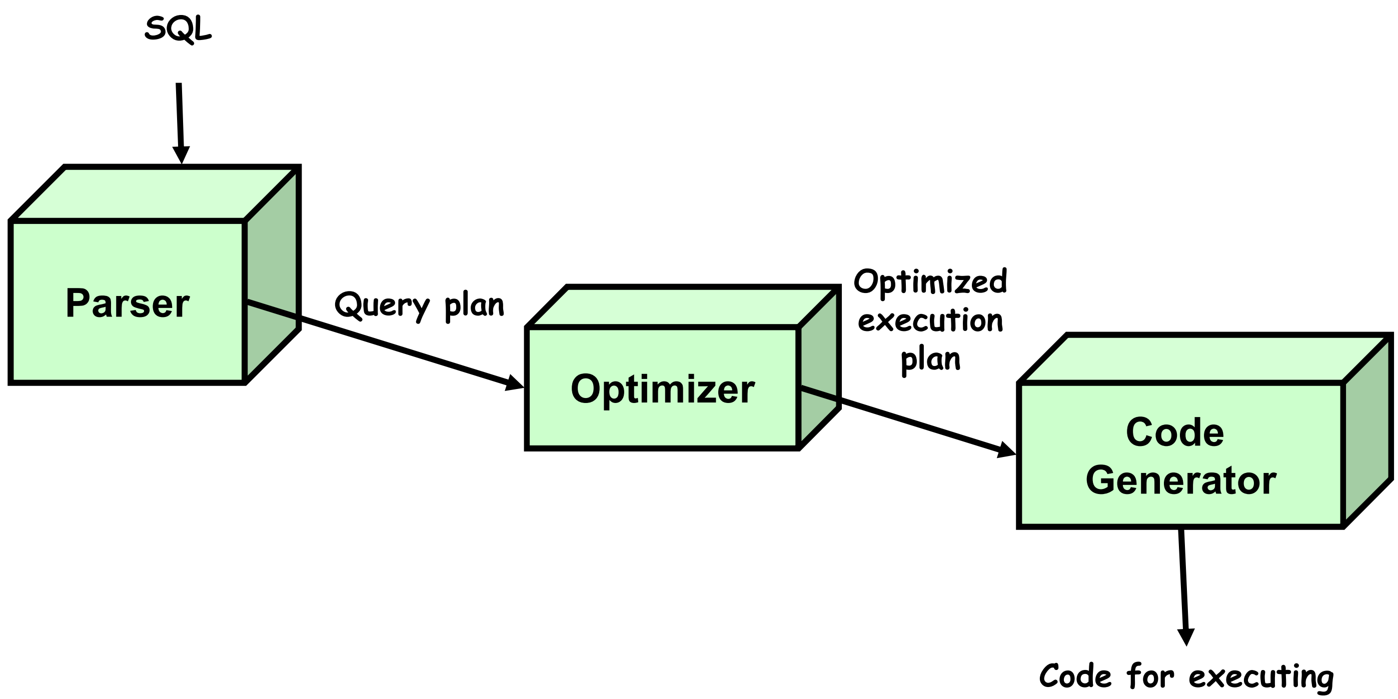
Lesson > Topic 1: Course overview

- Overview of query processing
- Query optimization

1.1. What is query processing

- The entire process or activities involved in retrieving data from the database
  - SQL query translation into low level instructions (usually relational algebra)
  - Query optimization to save resources, cost estimation or evaluation of query,
  - Query execution for the extraction of data from the database.

1.2. Phases of query processing



1.3. Parser

- Scans and parses the query into individual tokens and examines for the correctness of query
  - Does it containt the right keywords?
  - Does it conform to the syntax?
  - Does it containt the valid tables, attributes?
- Output: Query plan (Relational algebra)
- Input:
  - SELECT balance FROM account
    WHERE balance < 2500
- Output: Relational algebra (RA) expression
- But it’s not unique

$$\sigma_{balance<2500}(\Pi_{balance}(account))$$

$$\Pi_{balance}(\sigma_{balance<2500}(account))$$

1.4. Optimizer

- Input: RA expression

$$\Pi_{balance}(\sigma_{balance<2500}(account))$$

- Output: Query execution plan
  - Query execution plan = query plan + the algorithms for the executions of RA operations
  - Choose the cheapest execution plan out of the possible ones
    - Step 1: Equivalence transformation
    - Step 2: Annotation for the algorithm of the RA expression
    - Step 3: Cost estimation for different query execution plans

$$\Pi_{balance}$$

$$|$$

$$\sigma_{balance<2500}$$

use index 1

$$|$$

$$account$$

2. Understanding optimizer

- Choose the cheapest execution plan out of the possible ones
  - Step 1: Equivalence transformation
  - Step 2: Annotation for the algorithm of the RA expression
  - Step 3: Cost estimation for different query execution plans

2.1. Step 1: Equivalence transformation

- RA expressions are equivalent if they generate the same set of tuples on every database instance
- Equivalence rules:
  - Transform one relational algebra expression into equivalent one
  - Similar to numeric algebra: $a + b = b + a$, $a(b + c) = ab + ac$, etc.
- Why producing equivalent expressions?
  - equivalent algebraic expressions give the same result
  - but usually the execution time varies significantly

Equivalance tranformation rules

- (1) Conjunctive selection operations can be deconstructed into a sequence of individual seections; cascade of σ
  - $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$
- (2) Selection operations are commutative
  - $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
- (3) Only the final operations in a sequence of projection operations is needed; cascade of Π
  - $\Pi_{L_1}(\Pi_{L_2}(\ldots \Pi_{L_n}(E) \ldots)) = \Pi_{L_1}(E)$
- (4) Selections can be combined with Cartesian products and theta joins
  - $\sigma_{\theta_1}(E_1 \times E_2) = E_1 \bowtie_{\theta_1} E_2$
  - $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

- (5) Theta Join operations are commutative
  - $E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$
- (6) Natural join operations are associative
  - $E_1 \bowtie (E_2 \bowtie E_3) = (E_1 \bowtie E_2) \bowtie E_3$
  - Theta join are associative in the follwoing manner where θ_2 involves attributes from E_2 and E_3 only
  - $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$

- (7) Selection distributes over joins in the following ways
  - If predicate involves attributes of E_1 only
  - $\sigma_{\theta_1}\left(E_1 \bowtie_{\theta_2} E_2\right) = \sigma_{\theta_1}(E_1) \bowtie_{\theta_2} E_2$
  - If predicate θ_1 involves only attributes of E_1 and θ_2 involves only attributes of E_2 (a consequence of rule 7 and 1)
  - $\sigma_{\theta_1 \wedge \theta_2}\left(E_1 \bowtie_{\theta_3} E_2\right) = \sigma_{\theta_1}(E_1) \bowtie_{\theta_3} \sigma_{\theta_2}(E_2)$

- (8) Projection distributes over join as follows
  - $\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1}(E_1) \bowtie_\theta \Pi_{L_2}(E_2)$
  - If θ involves attributes in $L_1 \cup L_2$ only and L_i contains attributes of E_i
- (9) The set operations union and intersection are commutative
  - $E_1 \cup E_2 = E_2 \cup E_1$
  - $E_1 \cap E_2 = E_2 \cap E_1$
- (10) The union and intersection are associative
  - $(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$

- (11) The selection operation distributes over union, intersection, and set-difference
  - $\sigma_\theta(E_1 \cup E_2) = \sigma_\theta(E_1) \cup \sigma_\theta(E_2)$
  - $\sigma_\theta(E_1 \cap E_2) = \sigma_\theta(E_1) \cap \sigma_\theta(E_2)$
  - $\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - \sigma_\theta(E_2)$
- (12) The project operation distributes over the union
  - $\Pi_L(E_1 \cup E_2) = \Pi_L(E_1) \cup \Pi_L(E_2)$

2.2. Step 2: Execution algorithms of RA operations

- Algebra expression is not a query execution plan.
- Additional decisions required:
  - which indexes to use, for example, for joins and selects?
  - which algorithms to use, for example, sort-merge vs. hash join?
  - materialize intermediate results or pipeline them?

2.2.1. Basic Operators

- One-pass operators:
  - Scan
  - Select
  - Project
- Multi-pass operators:
  - Join
    - Various implementations
    - Handling of larger-than-memory sources
  - Semi-join
  - Aggregation, union, etc.

2.2.2. 1-Pass Operators: Scanning a Table

- Sequential scan: read through blocks of table
- Index scan: retrieve tuples in index order

2.2.3. Nested-loop JOIN

```
for each tuple tr in r {
  for each tuple ts in s {
    if (tr and ts satisfy the join condition θ) {
      add tuple tr × ts to the result set
    }
  }
}
```

- No index needed
- Any join condition types
- Expensive: $O(n^2)$

2.2.4. Single-loop JOIN (Index-based)

```
for each tube tr in R {
 ts = index.get(tuble s) {
         if ts.exist() {
                  add tr x ts to the result set
} } }
```

- Ony if there is at least one index on join attributes

2.2.5. Sort-merge JOIN

- Requires data physically sorted by join attributes

  Merge and join sorted files, reading sequentially a block at a time

  - Maintain two file pointers
    - While tuple at R < tuple at S, advance R (and vice versa)
    - While tuples match, output all possible pairings
  - Preserves sorted order of “outer” relation

✓ Very efficient for presorted data

✓ Can be “hybridized” with NL Join for range joins

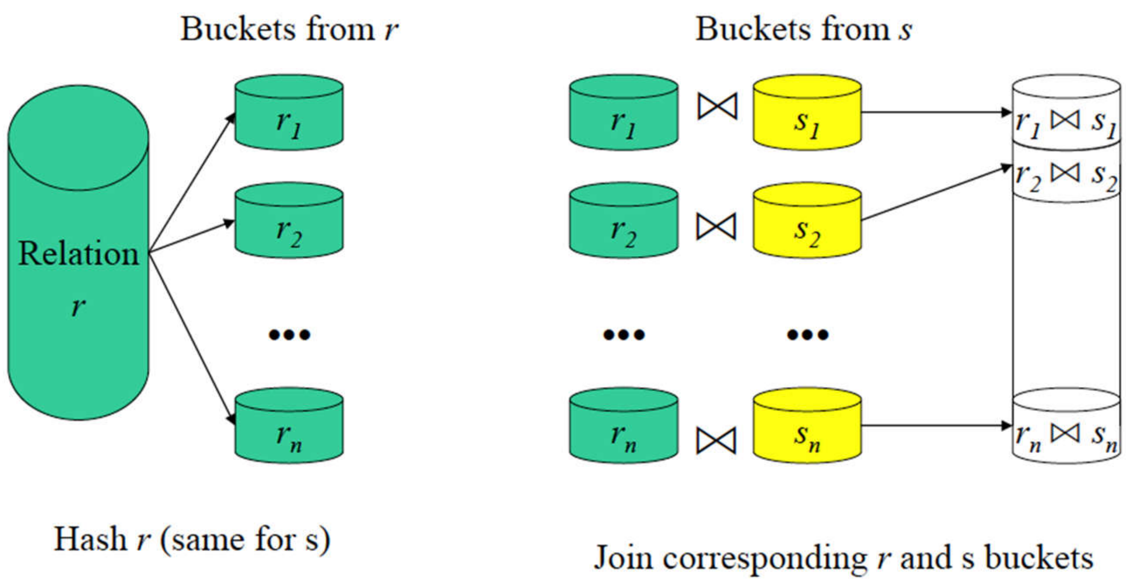
✗ May require a sort before (adds cost + delay)

- Cost: b(R) + b(S) plus sort costs, if necessary

  In practice, approximately linear, 3 (b(R) + b(S))

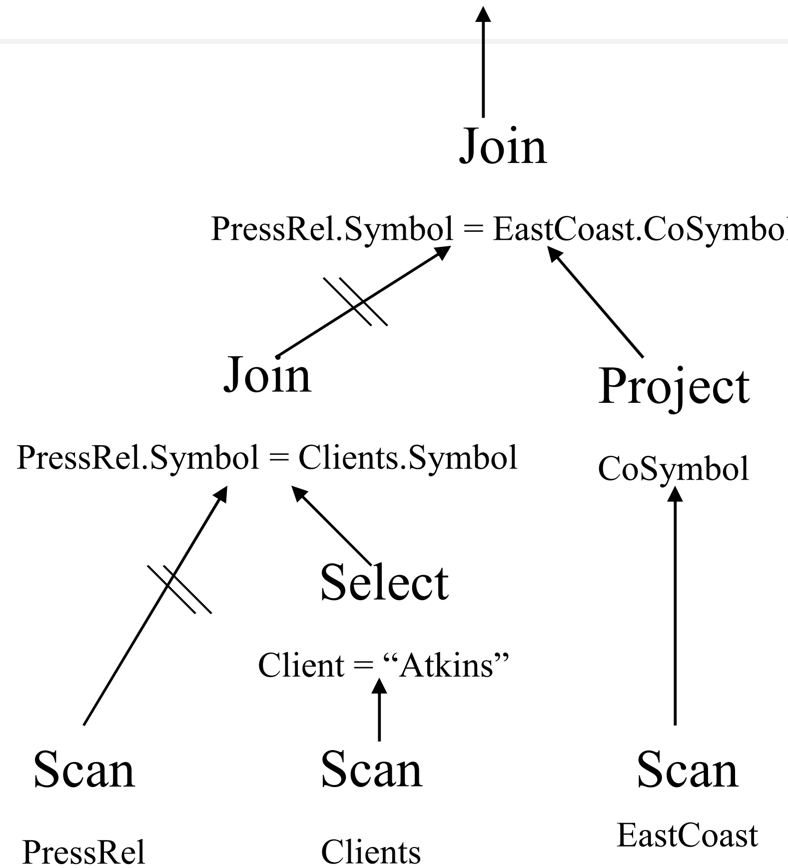
2.2.6. Partition-hash JOIN

- Hash two relations on join attributes
- Join buckets accordingly



2.2.7. Execution Strategy: Materialization vs. Pipelining

- Execution strategy defines how to walk the query execution plan
  - Materialization
  - Pipelining



Materialization

- Performs the innermost or leaf-level operations first of the query execution plan
- The intermediate result of each operation is materialized into temporary relation and becomes input for subsequent operations.
- The cost of materialization is the sum of the individual operations plus the cost of writing the intermediate results to disk
  - lots of temporary files, lots of I/O.

Pipelining

- Operations form a queue, and results are passed from one operation to another as they are calculated
- Pipelining restructures the individual operation algorithms so that they take streams of tuples as both input and output.
- Limitation
  - algorithms that require sorting can only use pipelining if the input is already sorted beforehand
  - since sorting by nature cannot be performed until all tuples to be sorted are known.

2.3. Step 3: Cost estimation

- Each relational algebra expression can result in many query execution plans
- Some query execution plans may be better than others
- Finding the fastest one
  - Just an estimation under certain assumptions
  - Huge number of query plans may exist

Cost estimation factors

- Catalog information: database maintains statistics about relations
- Ex.
  - number of tuples per relation
  - number of blocks on disk per relation
  - number of distinct values per attribute
  - histogram of values per attribute
- Problems
  - cost can only be estimated
  - updating statistics is expensive, thus they are often out of date

Choosing the cheapest query plan

- Problem: Estimating cost for all possible plans too expensive.
- Solutions:
  - pruning: stop early to evaluate a plan
  - heuristics: do not evaluate all plans
- Real databases use a combination of
  - Apply heuristics to choose promising query plans.
  - Choose cheapest plan among the promising plans using pruning.
- Examples of heuristics:
  - perform selections as early as possible
  - perform projections early avoid Cartesian products

Heuristic rules

- Query optimizers use the equivalence rules of relational algebra to improve the expected performance of a given query in *most cases*.
- The optimization is guided by the following **heuristics**:

(a) **Break apart conjunctive selections** into a sequence of simpler selections
(rule ①—preparatory step for (b)).

(b) **Move σ down the query tree** for the earliest possible execution
(rules ②, ⑦, ⑪—reduce number of tuples processed).

(c) **Replace σ–$\times$ pairs by $\bowtie$**
(rule ④ (a)—avoid large intermediate results).

(d) **Break apart and move as far down the tree as possible lists of projection attributes**, create new projections where possible
(rules ③, ⑧, ⑫—reduce tuple widths early).

(e) **Perform the joins with the smallest expected result first.**

Quiz

No	Question (Multiple Choice)	Answer (1,2,3,4)	Commentary
1	The reaction A → P follow the first-order rule. The half-life of the raction is 60 minutes. The time for A concentration of 12.5 % A. 60 min B. 30 min C. 120 min D. 180 min	D	3 times of half-life
2	Consider the combustion of propene: $2C_3H_6 + 9O_2 \rightarrow 6CO_2 + 6H_2O$. If the formation rate of CO_2 is 0.30 $M.s^{-1}$, the consumption rate of C_3H_6 is: A. 0.90 $M.s^{-1}$ B. 0.10 $M.s^{-1}$ C. 0.45 $M.s^{-1}$ D. 0.30 $M.s^{-1}$	A	Reaction rate: $\frac{1}{6}\frac{d[CO_2]}{dt} = -\frac{1}{2}\frac{d[C_3H_6]}{dt}$ Formation rate of CO_2: $R_{CO_3} = \frac{d[CO_2]}{dt}$ Consumption rate of C_3H_6 $R_{C_3H_6} = -\frac{d[C_3H_6]}{dt}$
3			

Outro > Summary

No	Topic	Summary
1	Introduction to Chemical Kinetics	- While the negative change in Gibbs energy (or Helmholtz free energy)
2		
3		
4		