COL362/632 Introduction to Database Management Systems

Query Processing – Scans/Selections and Projections

Kaustubh Beedkar

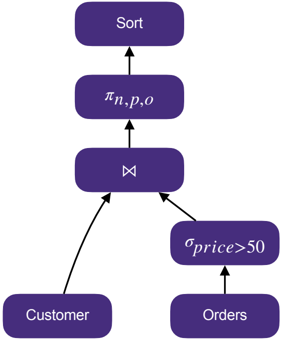
Department of Computer Science and Engineering
Indian Institute of Technology Delhi

Outline

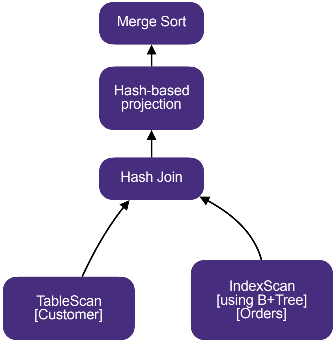
Query Processing (Overview)

- SQL : What?
- Physical plan: How?
- Query processing and optimization “compiles” a logical plan in to a physical plan

```
SELECT c.name, o.price, o.order_date
FROM customer c, orders o
WHERE c.customer_id = o.customer_id
AND o.price > 50
ORDER BY o.order_date DESC;
```

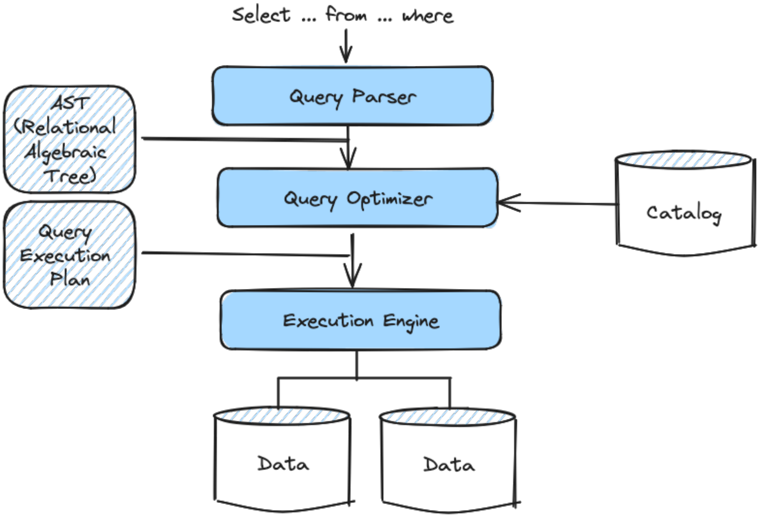


Logical Plan



Physical Plan

Query Processing (Overview)



Cost of an Operation

Cost depends on

- CPU cost depends on
  - # tuples
  - # index entries
  - # operators or functions
- I/O cost depends on
  - # blocks transferred by disk (b)
  - # random I/Os (S)

We usually ignore CPU cost (for disk-based systems)

- cost of an operation $= b \times t_T + S \times t_S$
- where, t_T is the time to transfer a block
- and t_S is the block access time (recall: seek time plus rotational delay)

Cost of an Operation

Assumptions and heuristics

- cost of accessing a random page $= \frac{1}{10}^{th}$ of the actual cost (why?)
- B+Tree nodes are all in memory, and using a B+Tree only incurs I/O cost for leaf node
- ...

Optimizer's goal is to minimize the **response time**

- Hard to estimate response time – depends on contents on the buffer, disks, data distribution
- Instead, minimize **resource consumption** (e.g., I/O cost)

Selections

- **Access path:** a "path" through which data can be located and accessed
- Scans are search algorithms that locate and retrieve "certain" records
    1. File scan
    2. Index scan

File Scan

Selection $(\sigma_{A=10})$

Linear search

- cost $= t_S + b_r \times t_T$

Linear search (if A is key)

- cost $= t_S + \frac{b_r}{2} \times t_T$ (average case)

Index Scan

Search algorithm that uses an index

- Use an index that is available on some predicate
- Cost depends on the index

Index Scan – B+Tree Index

Selection ($\sigma_{A=10}$)

Cost depends on height plus type of index

1. Clustering Index

   Case: A is key

   - cost $= (h_i + 1) \times (t_T + t_S)$

   Case: A is not a key

   - cost $= h_i \times (t_T + t_S) + t_S + (b \times t_T)$

Index Scan – B+Tree Index

Selection ($\sigma_{A=10}$)

Cost depends on height plus type of index

2. Non-clustering Index

   Case: A is key

   - cost $= (h_i + 1) \times (t_T + t_S)$

   Case: A is not a key

   - cost $= (h_i + n) \times (t_T + t_S)$

     $n =$ number of records fetched

Index Scan – B+Tree

Selection ($\sigma_{A>10}$ or $\sigma_{A\geq10}$)

1. Clustering Index
   - cost $= h_i \times (t_T + t_S) + t_S + b \times t_T$
   - Note: for $\sigma_{A<v}$ or $\sigma_{A\leq v}$, an index is not useful (Why?)

2. Non-clustering Index
   - cost $= (h_i + n) \times (t_T + t_S)$
   - Only use when n is small

     (HW: compare with Linear search. What would happen if n is large?)
   - Often used with Bitmaps and sorting (HW: Read about Bitmap Index Scan in PostgreSql)

General Selections

- So far we only looked at σ_p where $p \equiv A\ \theta\ v$
- Predicates often involve
  1. conjunctions, i.e., $\sigma_{p_1 \wedge p_2 \wedge \cdots \wedge p_n}$ e.g., $R.A = 10$ AND $R.B < 20$
  2. disjunctions, i.e., $\sigma_{p_1 \vee p_2 \vee \cdots \vee p_n}$ e.g., $R.A > 10$ OR $R.B < 20$

- General selections require **Index Matching**
  - an index matches a selection predicate if the index can evaluate it
  - e.g, `R(A,B,C)` and a hash index on composite key (A, B)

    $\sigma_{A=5 \wedge B=10}$ matches the index

    $\sigma_{A=5}$ **does not** matches the index

Index Matching – Hash Index

Selection $(\sigma_{p_1 \wedge p_2 \wedge \cdots \wedge p_n})$

A hash index on composite key (A,B,..) matches the selection condition if

- **all** attributes in the index search key appear in the predicate with $\theta \in \{=\}$

Example `R(A,B,C)`

selection condition	**hash index on (A,B)**	**hash index on (A)**
`A = 5 and B = 10`	yes	yes
`A = 5`	no	yes
`C = 5`	no	no
`A > 5 and B < 10`	no	no
`A = 5 and B = 10`	yes	yes
`A = 5 and B = 10 and C < 1`	yes	yes

Index Matching – B+Tree

Selection $(\sigma_{p_1 \wedge p_2 \wedge \cdots \wedge p_n})$

A B+Tree index on composite key (A,B,..) matches the selection condition if

- the attributes in the predicate form a prefix of the search key of B+Tree
- $\theta \in \{=, <, >, \leq, \geq\}$

Example `R(A,B,C)`

selection condition	B+Tree index on (A,B)	B+Tree index on (B,C)
`A = 5 and B = 10`	yes	yes
`A = 5`	yes	no
`B = 5`	no	yes
`A` $>$ `5 and B` $<$ `10`	yes	yes
`A = 5 and B = 10`	yes	yes
`A = 5 and B = 10 and C < 1`	yes	yes

Index Matching

When more than one index can match

Example

- B+Tree index on (A,B), and a hash index on (C)
- Selection condition: `A = 5 and B = 3 and C = 10`

Which index to use?

[Option 1] Use B+Tree index, retrieve tuples with (`A=5 and B=3`), then check `C=10` for every retrieved tuple

[Option 2] Use hash index, retrieve tuples with (`C=10`), then check `A=5 and B=3` for every retrieved tuple

[Option 3] Use B+Tree index, retrieve tuples with (`A=5 and B=3`), use hash index and retrieve tuples with `C=10`, intersect the record ids, only then fetch the tuples from disk

Disjunctive selection by union of identifiers

Example

- Hash index on (A) and B+Tree index on (B)
- Selection condition: `A = 5 or B > 10`

- [Option 1] Do a linear scan
- [Option 2] Use both indexes, fetch the records ids, and do a union, only then retrieve the tuples

Example

- Hash index on (A) + hash index on (B)
- Selection condition: `A = 5 or B > 10`

- [Option 1] Only option is to do a linear scan

Selectively Estimation

- Selectivity = fraction of tuples that need to be retrieved
- Goal should to chose the **most selective path** (Why?)
- But, estimating selectivity of an access path is a hard (active research area!)

Selectivity Estimation

Example

- hash index on (A, B)
- selection condition: `A=3 and B=6`

Selectivity $\approx \frac{1}{\#keys}$

- Number of keys can be determined from the index
- Only holds under uniform distribution assumption
- selection condition: `A=3 and B=6 and C=10`
- If no knowledge of #keys
  - selectivity = product of selectivity of primary conjuncts
  - or, use $\frac{1}{10}$ as default
  - holds only under independence assumption

Selectivity Estimation

Example

- Selection condition: $25 < \texttt{A} < 50$

$$\text{selectivity} \approx \frac{\text{interval}}{\max - \min}$$

Simple Projections

`select R.A, R.B, from ... where ...`

- Scan the file for each tuple output `R.A, R.B`

Projection w/ Duplicate Elimination

`select distinct R.A, R.B from ... where ...`

- first do simple projections
- remove duplicates
  - Sort-based duplicate elimination
    - Optimization: eliminate duplicates when merging runs
  - Hash-based duplicate elimination