

CSE462/562: Database Systems (Fall 24)

Lecture 11: Single-table query processing: Aggregation

10/1/2024

Recap on Single-Table QP

A few basic operators

Selection: σ

Projection: π (w/ and w/o deduplication)

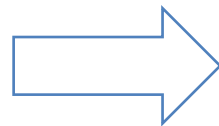
Aggregation: γ w/o or w/ group by

Set operators: $\cup, -, \cap$

Sorting

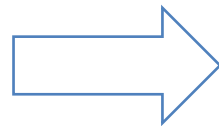
Cartesian product: \times or Join: \bowtie

```
SELECT E
FROM R
WHERE P
ORDER BY S
```



$Sort_S(\pi_E \sigma_P R)$

```
SELECT G, SUM(E)
FROM R
WHERE P
GROUP BY G
HAVING P'
ORDER BY S
```



$Sort_S(\sigma_{P'} \gamma_{SUM(E)} \sigma_P R)$

Measuring cost

- We'll start with the simplest single-table queries w/o or w/ aggregations
 - How to translate it into a query plan?
 - How to implement each operator?
 - How to measure the cost of each operator?
- For disk-based systems, we mainly measure the number of I/Os
 - Differences between random I/O and sequential I/O
 - Faster storage -> also need to measure the CPU cost
- A simple cost model
 - t_T : average time to transfer a page of data (data transfer time)
 - t_S : average time to randomly seek data (seek time + rotation delay)
 - For SSD, time overhead for initiating an I/O request
 - $\text{Cost} = B \times t_T + S \times t_S$
 - B : number of pages read/written; S : number of random I/O

Typical t_T and T_S

	HDD*	SSD†
t_T (ms)	0.1	0.01
t_S (ms)	4	0.09

Data from DB Concept book (Ch. 15.2).
Assuming 4KB pages.
* typical HDD with 40 MB/s transfer rate, 15000 rpm disk in 2018
† typical SATA SSD that supports 10K IOPS (QD-1), 400 MB/s sequential read rate

Measuring cost

- Other assumptions
 - Ignoring the buffer effect for random pages
 - Do consider the private workspace size M for the operators
 - Omitting the cost of transferring output to the user/disk
 - Common to any equivalent plan
- Notations: for relation R
 - T_R : number of records, N_R : number of pages in its heap file, B_R : (average) number of tuples per page
 - h_I : height of a B-tree index I over the file
 - M : private workspace size in pages
- Running example
 - $t_S = 4\text{ ms}$, $t_T = 0.1\text{ ms}$, 4000-byte page
 - Student: R(sid: int, name: varchar(19), login: varchar(19), major: char(2), adm_year: int)
 - 50 bytes/tuple, $B_R = 80$, $T_R = 40,000$, $N_R = 500$
 - Enrollment: E(sid: int, semester: char(3), cno: int, grade: double)
 - 20 bytes/tuple, $B_E = 200$, $T_E = 200,000$, $N_E = 1000$

Aggregation γ without grouping

*F is an aggregation function, e.g.,
SUM, COUNT, VAR, STDDEV, AVG, MIN, MAX or UDA etc.*

- $\gamma_{F_1(E_1), F_2(E_2), \dots, F_k(E_k)} Q$
 - Blocking
 - Only produce one row of output
- An aggregation can be expressed as three functions: $F = (F^{init}, F^{acc}, F^{final})$
 - Initialization $F^{init}: void \rightarrow A$ (where A is some internal state of the aggregation)
 - Accumulation $F^{acc}: (A, T) \rightarrow A$ or $(A, T) \rightarrow void$
 - Finalization $F^{final}: A \rightarrow V$ (where V is the final type of the aggregation)
 - Some systems also have an optional combine function $F^{combine}: (A, A) \rightarrow A$
 - allows parallelizing the aggregation
- Example: AVG of integers
 - $AVG^{init}()$: create a pair of (s, c) -- s : sum of values, c : number of values
 - $AVG^{acc}((s, c), x) = (s + x, c + 1)$
 - $AVG^{final}((s, c)) = 1.0 * s / c$
- Cost: does not add additional I/O cost

Aggregation γ without grouping

- Example: AVG of integers

- $AVG^{init}()$: create a pair of (s, c) -- s : sum of values, c : number of values
- $AVG^{acc}((s, c), x) = (s + x, c)$
- $AVG^{final}((s, c)) = 1.0 * s / c$

F is an aggregation function, e.g., SUM, COUNT, VAR, STDDEV, AVG, MIN, MAX or UDA etc.

- Consider a column in a table with the following values

- 5, 4, 1, 3, 2

- Steps:

- $AVG^{init}() = (0.0, 0)$
- $AVG^{acc}((0.0, 0), 5) = (5.0, 1)$
- $AVG^{acc}((5.0, 1), 4) = (9.0, 2)$
- $AVG^{acc}((9.0, 2), 1) = (10.0, 3)$
- $AVG^{acc}((10.0, 3), 3) = (13.0, 4)$
- $AVG^{acc}((13.0, 4), 2) = (15.0, 5)$
- $AVG^{final}((15.0, 5)) = 3.0 = \frac{5+4+1+3+2}{5}$

Aggregation in Project 3

- One of your tasks is to implement aggregation without grouping
 - Each aggregation “function” is
 - Denoted by `aggregation type` or `aggregation function name` in Taco-DB
 - Associated with three builtin functions (F^{init} , F^{acc} , F^{final})
 - Overloaded functions
 - Varies depending on operand types
 - Result type may also vary depending on input
 - To look up the aggregation, use `g_catcache->FindAggregationByXXX` functions.
 - See [code docs: CatCacheBase](#)
 - Identified by an `aggregation ID`
 - Not to be confused with `function IDs` of the init, acc, and final functions
 - Find the functions from [the catalog entry](#) for the aggregation
 - Should implement the catalog lookups and the logics in the preceding two slides.