CSE462/562: Database Systems (Fall 24)

Lecture 10: Single-table query processing: Selection, Projection & Expression Evaluation

9/26/2024

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Last updated: 10/9/2024 3:00 PM

Single-table queries

- 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?

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

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

SQL -> logical plan

- 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?

SELECT E
FROM R
WHERE P
ORDER BY S

$\Rightarrow$ $Sort_S(\pi_E \sigma_P R)$

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

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

Logical plan -> physical plan

- 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?

- 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 (later lectures)
  - Cartesian product: $\times$ or Join: $\bowtie$ (later lectures)

- Question: what are the alternatives? How to evaluate their efficiency?

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

  - Cost = $\mathrm{N_T} \times t_T + S \times t_S$
    - N_T: 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

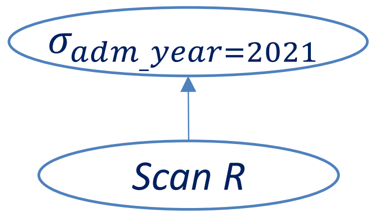
- Notations: for relation R
  - T_R: number of records, N_R: number of pages in its heap file, B_R: (average) number of tuples per page
  - h_I: height of a B-tree index I over the file
  - M: private workspace size in pages
- Running example
  - $t_S = 4\ ms$, $t_T = 0.1\ ms$, *4000-byte page*
  - Student: 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(<u>sid: int, semester: char(3), cno: int</u>, grade: double)
    - 20 bytes/tuple, $B_E = 200$, $T_E = 200{,}000$, $N_E = 1000$

Selection $\boldsymbol{\sigma}$

- Scan is usually the leaf-level of logical plans
  - Represents reading an entire relation -- not really a relational operator
- Selection $\sigma_P Q$
  - P is usually conjunctions or disjunctions $Q.attr\ op\ value$ but can also be User-Defined Functions (UDF)
  - selects records satisfying some predicate from the child
  - Child may be a scan or some other operators

  - Many possible implementation of selection depending on
    - the predicate P
    - the available file/index for the scan

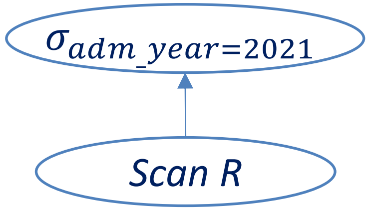
op is an operator: <, <=, =, <>, >, >=, …



Logical plan for $\sigma_{adm\_year=2021}R$

Simple selection: linear scan

- Consider a simple selection $\sigma_{R.attr\ op\ value}R$
  - Assume that the child is a relation stored in some disk file/index
- Most straight-forward implementation is linear scan
  - Scan each page and each record on the page
    - emits a record only if the predicate $R.attr\ op\ value$ evaluates to true



Logical plan for $\sigma_{adm\_year=2021}R$

  - Applies to any predicate P or file
  - Also works for pipelining -- can do selection on the fly without writing temporary files

- Cost: $t_S + N_R \times t_T$
  - 1 seek to the start of the file and N_R pages to read
  - the “last resort” -- usually the slowest implementation

  - cost for $\sigma_{adm\_year=2021}\ R$: $t_S + 500 \times t_T = 54\ ms$

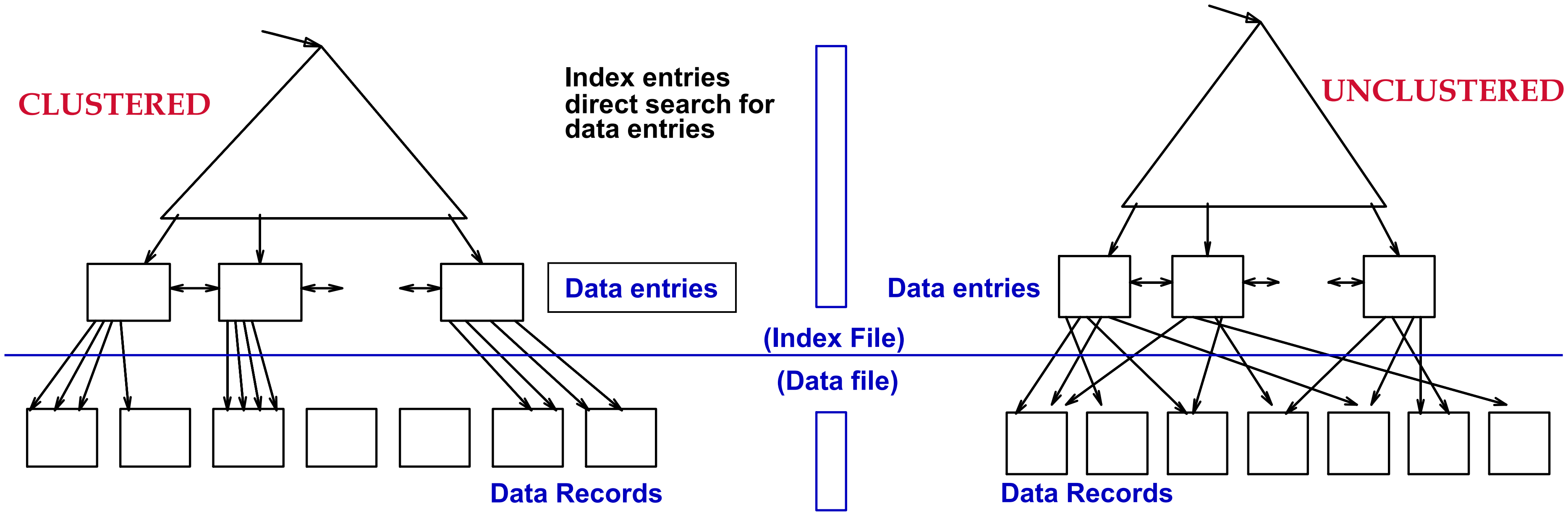
Simple selection: index scan

T: # of matching records
F: # of data entries per leaf page
N: # of pages with matching records

- If the file has an index I over the search key $k \in [K_{lo}, K_{hi}]$
- Assuming selectivity is $s = 0.1$, the number of matching records is T and the number of pages with matching records is N,
  cost =
  - Cost for finding qualifying data entries $I(N) = I_S(N)t_S + I_T(N)t_T$
    - $I_S(N)$: how many random accesses in the index before reaching the first data entry
    - $I_T(N)t_T$: how many pages in total were accessed, including those containing the data entries
  - + Cost for retrieving the heap records $H(N) = H_S(N,T)t_S + H_T(N,T)t_T$
    - $H_S(N,T)$: how many random accesses in the heap file
    - $H_T(N,T)$: how many pages in total were accessed in the heap file
  - Cost varies depending on the layout, selectivity of predicates and many other factors!

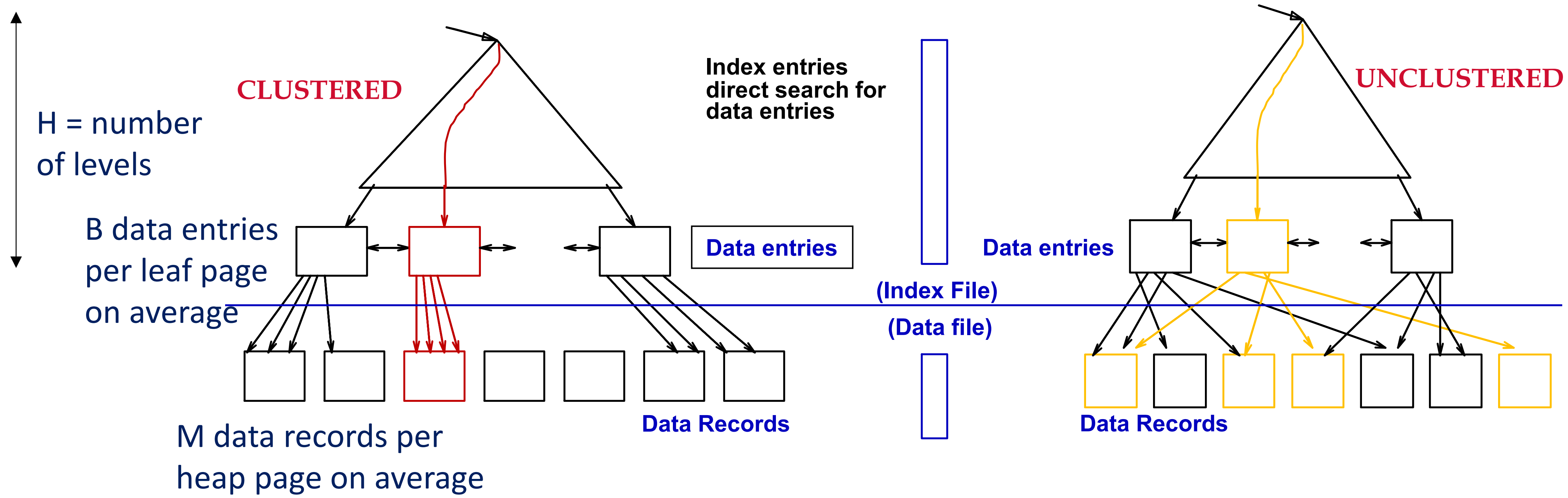
Clustered vs unclustered index

- Assuming data entries contain key and record id in the index (i.e., alternative 2).



Access cost of clustered vs unclustered index

- Assuming data entries contain key and record id in the index (i.e., alternative 2).
  - Cost of range scan with n matching data records in a B-Tree
    - assuming we ignore the buffer pool's effect
    - clustered: $H + \left\lceil \frac{n}{M} \right\rceil$ I/Os
    - unclustered: $H + \left\lceil \frac{n}{B} \right\rceil - 1 + n$ I/Os



General selection predicates

- Atom predicate: $attr\ op\ value$ or UDF
- General predicates:
  - Conjunction ∧ (and), disjunction ∨ (or), negation ¬ (not) of atoms or general predicates
  - e.g., $\sigma_{(adm\_year>=2019 \vee major='CS') \wedge sid >= 1000} R$
- Most general cases can always be handled by linear scans
  - Slow!
- Optimization for special cases:
  - Conjunction of simple selection predicates $\theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_r$
    - where θ_i is an atom
  - Disjunction of selection predicates $\theta_1 \vee \theta_2 \vee \cdots \vee \theta_r$
  - Transforming a predicate P into *Conjunctive Normal Form (CNF)* or *Disjunction Normal Form (DNF)* for additional optimization opportunities
    - e.g., $(adm\_year >= 2019 \vee major =' CS') \wedge sid >= 1000$ *(CNF)*
      $\Leftrightarrow (adm\_year >= 2019 \wedge sid \geq 1000) \vee (major =' CS' \wedge sid \geq 1000)$ *(DNF)*

Conjunctive selection with one index

- $\theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_r$
  - Choosing one or a prefix of predicates that can be answered using one index
    - Apply the rest of the predicates over the result on the fly
  - For instance, a B-Tree over (f_1, f_2) can select for predicates over a prefix of its index keys
    - $f_1\ op\ value$ (where $op \in \{<, \leq, =, >, \geq\}$)
    - $f_1 = value \wedge f_2\ op\ value$ (where $op \in \{<, \leq, =, >, \geq\}$)
    - If allow using skip scan (jump scan), f_2 op value or $f_1\ op\ value \wedge f_2\ op\ value$
  - What if there're multiple choices?
    - Considerations: selectivity, type of indexes, actual cost (access path selection in QO)

  - Cost is the same as index scans/bitmap index scans

Conjunctive selection with multiple indexes

- $\theta_1 \land \theta_2 \land \cdots \land \theta_r$
  - What if the atoms or several conjunctions of atoms can be answered by different indexes?
  - Example: $\sigma_{major='CS' \land adm\_year=2021} R$ when we have two indexes $I_1(major)$ and $I_2(adm\_year)$
- Algorithm:
  1. Collect all the RIDs using both indexes
  2. Compute the intersection of the RIDs
  3. Fetch the heap records of the RIDs in the result set
- Cost: index search + collecting data entries+ sort + intersection + fetching heap records

Partial matches for conjunctive selection

- $\theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_r$
  - What if only part of the predicates can be optimized with indexes
    - Apply the remaining predicates over the result and discard those that do not satisfy
    - e.g., $\sigma_{major='CS' \wedge adm\_year=2021}$ with a hash index $I(major)$
      - Index Scan for all CS majors using $I(major)$
      - Apply the predicate $adm\_year = 2021$ over the heap records on the fly

    - Note the remaining predicates do not need to be in conjunctive normal form!
      - Can be arbitrary predicates (e.g., UDF)

Disjunction selection with multiple indexes

- $\theta_1 \vee \theta_2 \vee \cdots \vee \theta_r$
  - Only optimizable if all clauses θ_i can be optimized using some index
  - Otherwise, fall back to linear scan
- Algorithm:
  1. Collect all the RIDs using both indexes
  2. Compute the union of the RIDs
  3. Fetch the heap records of the RIDs in the result set
- Cost: index search + collecting data entries+ sort + union + fetching heap records

An excursion: expression evaluation

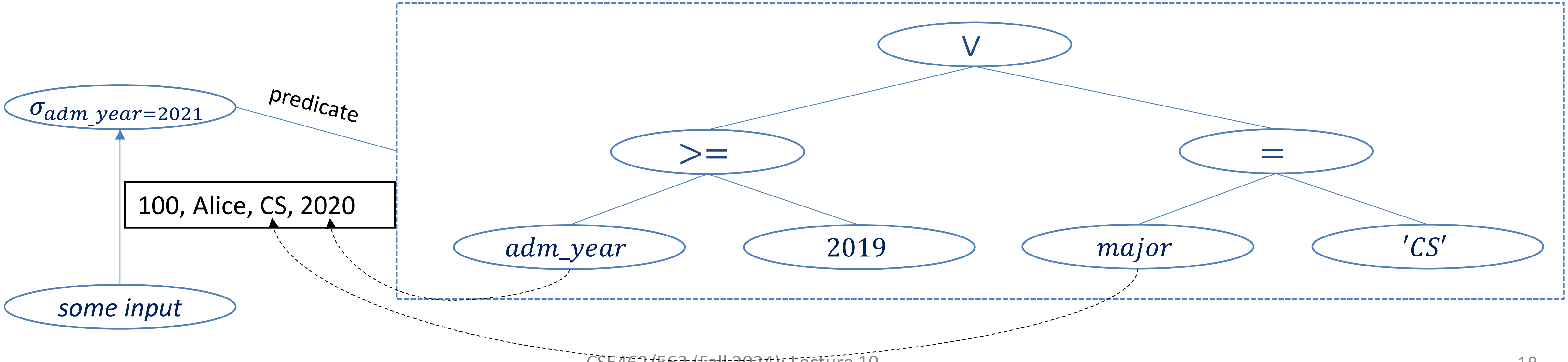
- So far, we assume expression evaluation is a black box
  - Does the predicate evaluate to true in selection?
  - Projection list evaluation?
  - …

- How does it work?
  - How costly are they?

Expression tree

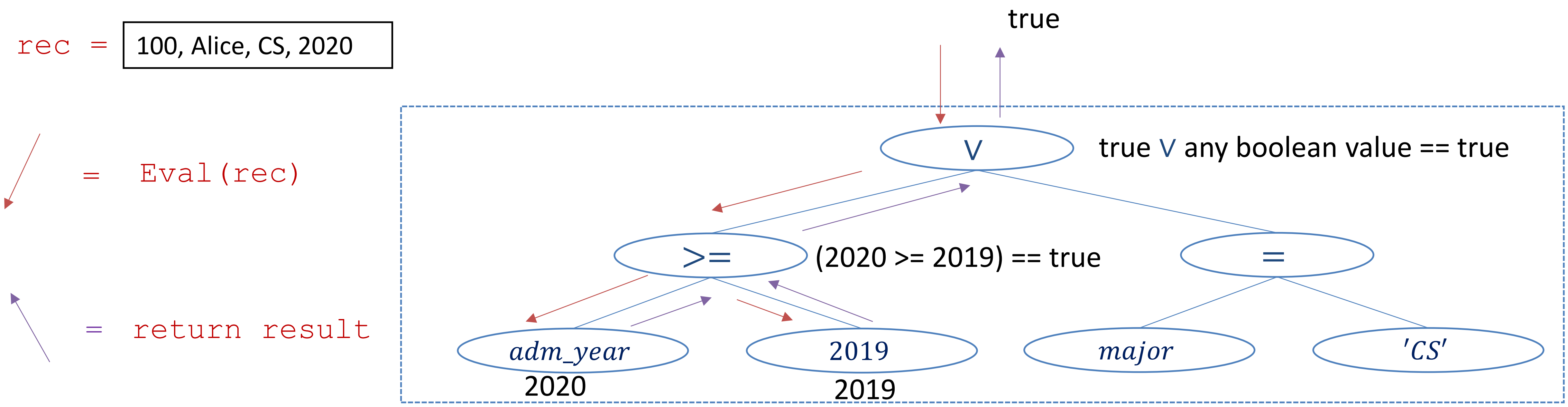
- A tree that represents an expression
  - Leaf nodes: literals, variables
  - Internal nodes: operators (+, -, *, /, …), function calls, …
- Expressions in QP are attached to a plan node
  - Variables refers to columns in the output of some plan node
    - usually output from child, but could be intermediate outputs within certain operators
- Example: predicate $adm\_year >= 2019 \lor major = 'CS'$

Q: what are the variables in query plan?
A: (short answer) columns in the output



Expression evaluation

- Interpretation vs Compilation
  - type checking?
- In the course project Taco-DB, we use interpretation (for ease of implementation)
  - recursive evaluation through `Eval()` calls



Projection π

- Without deduplication
  - evaluate projection list for the records on the fly
  - cost: no additional I/O
  - sometimes baked into other operators (i.e., all operators can be followed by an implicit projection)

- With deduplication
  - Requires materialization (blocking)
  - Hash or Sort
    - Hash -> build a hash table where duplicates are dropped
    - Sort -> emit a record only if it is the first record or it is different from the previous one
  - Result set fits in memory => easy to implement (does not add I/O cost)
  - When result sets exceed configured workspace size M,
    - Need to use external hashing and sorting algorithms (next lecture)
    - Optimization opportunities
    - Will come back to this later after we discuss external hashing and sorting