CSE462/562: Database Systems (Spring 23)
Lecture 18: Query Optimization Overview
4/20/2023
**Query processing overview**

**ODBC/JDBC/ command line frontend**

**SQL Query**
```
SELECT S.name, E.grade
FROM student S, enrollment E
WHERE S.sid = E.sid
    AND S.adm_year = 2021
    AND E.cno = 562;
```

**SQL Parser**

(Extended) Relational Algebra
```
\pi_{S.name,E.grade} \sigma_{S.adm\_year=2021 \land E.cno=562} S \bowtie_{S.sid=E.sid} E
```

**Query Optimizer**

**Physical plan**
```
\pi_{S.name,E.grade}
\sigma_{S.adm\_year=2021 \land E.cno=562}
S \bowtie_{S.sid=E.sid}

\pi_{S.name,S.sid}
\pi_{S.name,E.grade}
Index Scan
student S
Index Scan
enrollment E
Index Nested Loop Join
```

**Logical plan**
```
\pi_{S.name,E.grade}
\sigma_{S.adm\_year=2021 \land E.cno=562}
S \bowtie_{S.sid=E.sid}
Scan
student S
Scan
enrollment E
```

**Query result**
```
S.name | E.grade
Alice  | 4.0
Charlie| 2.3
```

2 rows
Query optimization overview

- Query can be converted to relational algebra
- Relational Algebra converted to tree, joins as branches
- Each operator has implementation choices
- Operators can also be applied in different order!

```sql
SELECT R.name
FROM Enroll E, Students R
WHERE E.sid = R.sid AND E.cno >= 500 AND R.adm_year = 2020
```
Query optimization overview

- **Plan**: Tree of R.A. ops (and some others) with choice of algorithm for each op.
  - Each operator typically implemented using a `pull’ interface: when an operator is `pulled’ for the next output tuples, it `pulls’ on its inputs and computes them.

- Two main issues:
  - For a given query, what plans are considered?
    - Algorithm to search plan space for cheapest (estimated) plan.
  - How is the cost of a plan estimated?

- Ideally: Want to find best plan.
- Reality: Avoid worst plans!

Relational operators have a uniform *iterator* interface:

```
open( ), get_next( ), close( )
```
Cost-based query optimizer

Usually there is a heuristics-based rewriting step before the cost-based steps.

```
Select *
From Blah B
Where B.blah = blah
```
Running example

• 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
  
  • Student: $R(sid: \text{int}, \text{name: varchar(19), login: varchar(19), major: char(2), adm_year: int})$
    
    • 50 bytes/tuple, $B_R = 80$, $T_R = 40,000$, $N_R = 500$
    
    • Assume the student records in the table span 10 years (between 2012 and 2022)
  
  • Enrollment: $E(sid: \text{int}, \text{semester: char(4), cno: int, grade: double})$
    
    • 20 bytes/tuple, $B_E = 200$, $T_E = 200,000$, $N_E = 1000$
    
    • Assume 50% of the enrollment records belong to the graduate level ($\geq 500$) courses

• Consider a simplified cost model: $cost = \#\text{page}\_\text{transfers}$ (i.e., ignoring the random seeks)
  
  • Often good enough for approximating the trend of the cost relative to data size
  
  • Correct size estimation is key to a correct comparison of costs

• Assume we have 5 pages in the buffer
Motivating example

```
SELECT R.name
FROM Enroll E, Students R
WHERE E.sid=R.sid AND
  E.cno=562 AND R.adm_year = 2020
```

- By no means the worst plan!
- Misses several opportunities: selections could have been `pushed' earlier, no use is made of any available indexes, etc.
- **Goal of optimization**: To find more efficient plans that compute the same answer.

Cost = 1000 + 1000 * 500 = 501,000 I/Os
Relational algebra equivalence

- Rules that allow the optimizer to transform a logical plan into an equivalent plan with the *same* output over any database instance

**Selections:**
- Cascade: $\sigma_{\theta_1 \land \theta_2} E \equiv \sigma_{\theta_1} \sigma_{\theta_2} E$
- Commutative: $\sigma_{\theta_1} \sigma_{\theta_2} E \equiv \sigma_{\theta_2} \sigma_{\theta_1} E$

**Projections:**
- Cascade: $\pi_{A_1} \pi_{A_2} \ldots \pi_{A_n} E \equiv \pi_{A_1}(E)$ where $A_1 \subseteq A_2 \subseteq \ldots \subseteq A_n$
  - Only need to perform the final projection in a sequence of projections

**(Inner) Joins or Cartesian product:**
- Commutative: $E_1 \bowtie_\theta E_2 \equiv E_2 \bowtie_\theta E_1$ (allows switching the inner and outer)
- Associative
  - Special case natural join: $(E_1 \bowtie E_2) \bowtie E_3 \equiv E_1 \bowtie (E_2 \bowtie E_3)$
  - General theta join: $(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \land \theta_3} E_3 \equiv E_1 \bowtie_{\theta_1 \land \theta_3} (E_2 \bowtie_{\theta_2} E_3)$
- Implication: inner joins can be done in any order!
  - **Join reordering**: an important optimization step in DBMS

Assuming $\theta_2$ only involves fields in $E_2$ and $E_3$
Relational algebra equivalence

• Rules for more than one operator
  
  • *Selection can be combined with inner join/cartesian product*
    \[
    \sigma_{\theta_1} (E_1 \bowtie_{\theta_2} E_2) \equiv E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2
    \]

  • **Projection push-down:** select/join and projection commutes (provided that the predicate only involves the projected fields)
    \[
    \pi_A \sigma_{\theta} E \equiv \sigma_{\theta} \pi_A E \quad \text{when } \text{Var}(\theta) \subseteq A
    \]
    \[
    \pi_{A_1 \cup A_2} (E_1 \bowtie_{\theta} E_2) \equiv \pi_{A_1} E_1 \bowtie_{\theta} \pi_{A_2} E_2 \quad \text{when } \text{Var}(\theta) \subseteq A_1 \cup A_2 \text{ and } A_1, A_2 \text{ only involve fields from } E_1, E_2, \text{ resp.}
    \]

  • **Selection push-down:** join and select commutes (provided that the selection predicate only involves attributes from one side)
    \[
    \sigma_{\theta_1} (E_1 \bowtie_{\theta} E_2) \equiv (\sigma_{\theta_1} E_1) \bowtie_{\theta} E_2 \quad \text{when } \text{Var}(\theta_1) \subseteq A(E_1) \text{ (set of fields in } E_1)\]

• More rules about other operators, e.g., aggregation, set operations, sort, ...

• Note: rules involving outer joins may be different
  
  • Exercise: Can we always push selection through outer joins? What about projections?
Selection push-down (no index)

- Heuristics 1: perform selections as early as possible
  - Selection is often very cheap or “free” (in I/O only cost model)
  - reduces intermediate size

Cost = \( 501,000 \) I/Os

Collect one page from the outer plan, rather than the underlying scan.

Cost = \( 1000 + [1000 \times 0.5] \times 500 = 251,000 \) I/Os
Selection push-down (no index)

- Can also push-down on the other side

\[ \pi_{R.name} \]

\( \sigma_{R.adm\_year=2020} \) (On-the-fly)

\( \sigma_{E.cno\geq 500} \) (On-the-fly)

\( \bowtie_{E.sid=R.sid} \) (Block nested loop)

Students R

Enroll E

Cost = 251,000 I/Os

\( \sigma_{R.adm\_year=2020} \) (On-the-fly)

\( \sigma_{E.cno\geq 500} \) (On-the-fly)

\( \bowtie_{E.sid=R.sid} \) (Block nested loop)

Students R

Enroll E

Cost = 251,000 I/Os

No impact on I/O because BNL scans the inner plan once for every outer block.
Join reordering

- Different join ordering may result in different cost
  - even if we use the same join algorithm
  - Generally, the outer plan should have a smaller output in BNL
    - what about hash join/sort merge join?

\[
\begin{align*}
\text{Enroll } E & \quad \bowtie_{E.sid=R.sid}^{\pi_{R.name}} (\text{Block nested loop}) \quad \sigma_{E.cno\geq500}^{(\text{On-the-fly})} \\
\text{Students } R & \quad \sigma_{R.adm\_year=2020}^{(\text{On-the-fly})} \\
\text{Cost} & = 251,000 \text{ I/Os}
\end{align*}
\]

\[
\begin{align*}
\text{Students } R & \quad \sigma_{R.adm\_year=2020}^{(\text{On-the-fly})} \\
\text{Enroll } E & \quad \bowtie_{E.sid=R.sid}^{\pi_{R.name}} (\text{Block nested loop}) \quad \sigma_{E.cno\geq500}^{(\text{On-the-fly})} \\
\text{Cost} & = 500 + [500 \times 0.1] \times 1000 = 50,500 \text{ I/Os}
\end{align*}
\]
Materialization of inner plan

• We can also choose to materialize the inner plan for BNL to save repeated scan on the original relation.
Materialization of inner plan

• Sometimes with materialization, it might be cheaper to use the larger plan as the outer

\[
\begin{align*}
\sigma_{R.adm\_year=2020} & \quad (\text{On-the-fly}) \\
\pi_{R.name} & \quad (\text{On-the-fly}) \\
\sigma_{E.cno \geq 500} & \quad (\text{materialize in temporary file}) \\
\bowtie_{E.sid=R.sid} & \quad (\text{Block nested loop}) \\
\end{align*}
\]

Students R

\[
\begin{align*}
\text{Enroll E} & \quad (\text{On-the-fly}) \\
\end{align*}
\]

\[
\begin{align*}
\text{Cost} &= 1000 + [1000 \times 0.5] + 500 + [500 \times 0.1] \times [1000 \times 0.5] \\
&= 27,000 \text{ I/Os}
\end{align*}
\]

\[
\begin{align*}
\sigma_{R.adm\_year=2020} & \quad (\text{On-the-fly}) \\
\pi_{R.name} & \quad (\text{On-the-fly}) \\
\sigma_{E.cno \geq 500} & \quad (\text{materialize in temporary file}) \\
\bowtie_{E.sid=R.sid} & \quad (\text{Block nested loop}) \\
\end{align*}
\]

Students R

\[
\begin{align*}
\text{Enroll E} & \quad (\text{On-the-fly}) \\
\end{align*}
\]

\[
\begin{align*}
\text{Cost} &= 500 + [500 \times 0.1] + 1000 + [1000 \times 0.5] \times [500 \times 0.1] \\
&= 26,550 \text{ I/Os}
\end{align*}
\]
Projection push-down

- Heuristics 2: apply projection as early as possible
  - helps if materializing plan output

Enrollment: $E(\text{sid: int, semester: char(3), cno: int, grade: double})$

$20 \text{ bytes/tuple} \Rightarrow \pi_{E.\text{sid}} : \frac{4}{20} = 20\% \text{ in size after projection}$

\[
\begin{align*}
\text{Cost} &= 1000 + [1000 \times 0.5] + 500 + [500 \times 0.1] \times [1000 \times 0.5] \\
&= 27,000 \text{ I/Os}
\end{align*}
\]
Projection push-down

• More projection push-down on the other side

\[ R(\text{sid: int, name: varchar(19), login: varchar(19), major: char(2), adm_year: int}) \]

50 bytes/tuple => \[ \pi_{R.name,R.sid} : \frac{4+19+1}{50} = 48\% \text{ -- assuming VARCHAR uses ‘\0’ at the end} \]

\[ \sigma_{R.adm\_year=2020} \]

\[ \pi_{E.sid} \]

\( \bowtie_{E.sid=R.sid} \)

(On-the-fly)

(On-the-fly)

(Block nested loop)

(Block nested loop)

Students R

(On-the-fly)

(On-the-fly)

Enroll E

Cost = 1000 + [1000 * 0.5 * 0.2]
+ 500 + [500 * 0.1] * [1000 * 0.5 * 0.2]
= 6,600 I/Os

Cost = 1000 + [1000 * 0.5 * 0.2]
+ 500 + [500 * 0.1 * 0.48] * [1000 * 0.5 * 0.2]
= 4,000 I/Os

50 bytes/tuple => \[ \pi_{R.name,R.sid} : \frac{4+19+1}{50} = 48\% \text{ -- assuming VARCHAR uses ‘\0’ at the end} \]
Choice of join algorithms

- If we switch to sort-merge join with 5 buffers

\[
\begin{align*}
\text{Students } R & \quad \text{Enroll } E \\
\pi_{R.\text{name}} & \quad \text{(On-the-fly)} \\
\bowtie_{E.\text{sid}=R.\text{sid}} & \quad \text{(Block nested loop)} \\
\pi_{E.\text{sid}} & \quad \text{(materialize in temporary file)} \\
\sigma_{R.\text{adm\_year}=2020} & \quad \text{(On-the-fly)} \\
\sigma_{E.\text{cno} \geq 500} & \quad \text{(On-the-fly)} \\
\pi_{R.\text{name},R.\text{sid}} & \quad \text{(Materialization)} \\
\text{Students } R & \quad \text{Enroll } E \\
\pi_{E.\text{sid}} & \quad \text{(On-the-fly)} \\
\sigma_{R.\text{adm\_year}=2020} & \quad \text{(On-the-fly)} \\
\sigma_{E.\text{cno} \geq 500} & \quad \text{(On-the-fly)} \\
\pi_{R.\text{name},R.\text{sid}} & \quad \text{(Materialization)} \\
\text{Students } R & \quad \text{Enroll } E \\
\end{align*}
\]

Cost = \( 1000 + [1000 \times 0.5 \times 0.2] + 500 + [500 \times 0.1 \times 0.48] \times [1000 \times 0.5 \times 0.2] \)

\[= 4,000 \text{ I/Os} \]

Cost = ?
Choice of join algorithms

- Sort outer:
  - Size after pass 0: \([500 \times 0.1 \times 0.48] = 24\)
    - 4 pages/run, 6 runs
    (need one input buffer for table scan)
  - # merge passes = \([\log_4 6]\) = 2
  - Total I/O: \(500 + 24 + 2 \times 2 \times 24 = 620\)

- Sort inner: # I/O = 1700

- Merge
  - assuming \(d = 5\) and always fit in one page
  - \(24 + 100 = 124\)

- Total cost = \(620 + 1700 + 124 = 2,444\) I/Os
  - vs BNL: 4,000 I/Os

Cost = ?
Using indexes

- If we have a clustered B-Tree index over \( R(adm\_year) \), \( h = 3 \)

\[
\begin{align*}
\text{Cost} &= 1000 + [1000 \times 0.5 \times 0.2] \\
&+3 + [500 \times 0.1 \times 0.48] \\
&+[500 \times 0.1 \times 0.48] \times [1000 \times 0.5 \times 0.2] \\
&= 3,527 \text{ I/Os}
\end{align*}
\]
Using indexes

- If we have an unclustered B-Tree index over \( E(sid) \), \( h = 3 \)
  - Generally, index nested loop is a bad choice unless both of the following is true
    - outer plan output size is small
    - join is very selective

\[
\begin{align*}
\text{Cost} &= 3 + [500 \times 0.1 \times 0.48] + [40000 \times 0.1] \times (3 + 5) \\
&= 32,027 \text{ I/Os} \quad \text{(vs 3,527 I/Os with BNL!)}
\end{align*}
\]

Assuming each student has 5 enrollment records on average.
What’s needed for query optimization?

• A closed set of operators
  • Relational ops (table in, table out)
  • Encapsulation based on iterators

• Plan space, based on
  • Based on relational equivalences

• Cost Estimation, based on
  • Cost formulas
  • Size estimation, based on
    • Catalog information on base tables
    • Selectivity (Reduction Factor) estimation

• A search algorithm
  • To sift through the plan space based on cost!
Summary

• Today’s lecture
  • Query optimization overview
  • Relational algebra equivalence
  • Query optimization is needed to ensure not-too-bad performance if not the best
    • Need to understand the impact of cost model/physical data layout/indexing for a given query

• Next lecture(s)
  • Plan size and cost estimation
  • How to search in the optimization space
    • System R style query optimizer