Query Optimization

- Remember that SQL is declarative.
  - User tells the DBMS what answer they want, not how to get the answer.
- There can be a big difference in performance based on plan is used:
  - See last week: 5.7 days vs. 45 seconds

1970s – Relational Model

- Ted Codd saw the maintenance overhead for IMS/Codasyl.
- Proposed database abstraction based on relations:
  - Store database in simple data structures.
  - Access it through high-level language.
  - Physical storage left up to implementation.
IBM System R

- Skunkworks project at IBM Research in San Jose to implement Codd’s ideas.
- Had to figure out all of the things that we are discussing in this course themselves.
- IBM never commercialized System R.

First implementation of a query optimizer.
- People argued that the DBMS could never choose a query plan better than what a human could write.
- A lot of the concepts from System R’s optimizer are still used today.

Today’s Class

- History & Background
- Relational Algebra Equivalences
- Plan Cost Estimation
- Plan Enumeration
- Nested Sub-queries

Relational Algebra Equivalences

- A query can be expressed in different ways.
- The optimizer considers variations and choose the one with the lowest cost.
- How do we know whether two queries are equivalent?
Relational Algebra Equivalences

- Two relational algebra expressions are equivalent if they generate the same set of tuples.

\[ \pi_{\text{name, cid}}(\sigma_{\text{grade} = 'A'}(\sigma_{\text{sid} = \text{enrolled.sid}}(\sigma_{\text{enrolled.grade} = 'A'}(\text{student} \bowtie \text{enrolled})))) \]

\[ = \]

\[ \pi_{\text{name, cid}}(\text{student} \bowtie (\sigma_{\text{grade} = 'A'}(\text{enrolled}))) \]

Predicate Pushdown

\[ \text{SELECT name, cid} \]
\[ \text{FROM student, enrolled} \]
\[ \text{WHERE student.sid = enrolled.sid AND enrolled.grade = 'A'} \]

Relational Algebra Equivalences

- Selections:
  - Perform them early
  - Break a complex predicate, and push down
  \[ \sigma_{p1 \land p2 \land \ldots \land pn}(R) = \sigma_{p1}(\sigma_{p2}(\sigma_{\ldots pn}(R))...) \]
  - Simplify a complex predicate
  \[ (X=Y \ \text{AND} \ Y=3) \rightarrow X=3 \ \text{AND} \ Y=3 \]
Relational Algebra Equivalences

- **Projections:**
  - Perform them early
    - Smaller tuples
    - Fewer tuples (if duplicates are eliminated)
  - Project out all attributes except the ones requested or required (e.g., joining attr.)

- *This is not important for a column store...*

Projection Pushdown

```
SELECT name, cid
FROM student, enrolled
WHERE student.sid = enrolled.sid
AND enrolled.grade = 'A'
```

- **Joins:**
  - Commutative, associative
    
    \[
    R \bowtie S = S \bowtie R
    \]
    
    \[
    (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)
    \]

- Q: How many different orderings are there for an $n$-way join?
Relational Algebra Equivalences

• **Joins:** How many different orderings are there for an n-way join?
  - A: [Catalan number](https://en.wikipedia.org/wiki/Catalan_number) \( \sim 4^n \)
    - Exhaustive enumeration: too slow.
  - We’ll see in a second how an optimizer limits the search space...

Today’s Class

• History & Background
• Relational Algebra Equivalences
• Plan Cost Estimation
• Plan Enumeration

Cost Estimation

• How long will a query take?
  - **CPU**: Small cost; tough to estimate
  - **Disk**: # of block transfers
  - **Memory**: Amount of DRAM used
  - **Network**: # of messages
• How many tuples will be read/written?
• What statistics do we need to keep?

Cost Estimation – Statistics

• For each relation \( R \) we keep:
  - \( N_R \rightarrow \) # tuples
  - \( S_R \rightarrow \) size of tuple in bytes
  - \( V(A,R) \rightarrow \) # of distinct values of attribute ‘A’

\[
\begin{array}{c|c|c|c}
\#1 & \#2 & \#3 & \#N_R \\
\hline
\hline
\end{array}
\]

Faloutsos/Pavlo  CMU SCS 15-415/615  16
Faloutsos/Pavlo  CMU SCS 15-415/615  17
Faloutsos/Pavlo  CMU SCS 15-415/615  18
Faloutsos/Pavlo  CMU SCS 15-415/615  19
Derivable Statistics

- $F_R \rightarrow \text{max# records/block}$
- $B_R \rightarrow \# \text{blocks}$
- $SC(A,R) \rightarrow \text{selection cardinality}$
  \[ \text{avg# of records with } A=\text{given} \]

Faloutsos/Pavlo CMU SCS 15-415/615 20

Additional Statistics

- For index $i$:
  - $F_i \rightarrow \text{average fanout (~50-100)}$
  - $HT_i \rightarrow \# \text{levels of index } i (~2-3)$
    \[ \sim \log(\#\text{entries})/\log(F_i) \]
  - $LB_i \# \rightarrow \text{blocks at leaf level}$

Faloutsos/Pavlo CMU SCS 15-415/615 22

Statistics

- Where do we store them?
- How often do we update them?
- Manual invocations:
  - Postgres/SQLite: ANALYZE
  - MySQL: ANALYZE TABLE

Faloutsos/Pavlo CMU SCS 15-415/615 23
Selection Statistics

- We saw simple predicates (name=“Kayne”)
- How about more complex predicates, like
  - salary > 10000
  - age=30 AND jobTitle=“Costermonger”
- What is their selectivity?

Selections – Complex Predicates

- Selectivity \( sel(P) \) of predicate \( P \):
  \( \text{== fraction of tuples that qualify} \)
- Formula depends on type of predicate.
  - Equality
  - Range
  - Negation
  - Conjunction
  - Disjunction

Assume that \( V(\text{rating, sailors}) \) has 5 distinct values (0–4) and \( N_R = 5 \)

**Equality Predicate**: \( A=\text{constant} \)

- \( sel(A=\text{constant}) = \frac{SC(P)}{V(A,R)} \)
- Example: \( sel(\text{rating}=‘2’) = \)
• Assume that $V(\text{rating, sailors})$ has 5 distinct values (0–4) and $N_R = 5$

• Equality Predicate: $A=\text{constant}$
  – $\text{sel}(A=\text{constant}) = SC(P) / V(A,R)$
  – Example: $\text{sel}(\text{rating}='2') =$

\[
\begin{array}{c|c|c|c|c|c}
\text{rating} & 0 & 1 & 2 & 3 & 4 \\
\hline
\text{count} & \text{bar} & \text{bar} & \text{bar} & \text{bar} & \text{bar} \\
\end{array}
\]

\[
\text{SC(rating}='2')=1
\]

\[
V(\text{rating,R})=5
\]

\[
\begin{array}{c|c|c|c|c|c}
\text{rating} & 0 & 1 & 2 & 3 & 4 \\
\hline
\text{count} & \text{bar} & \text{bar} & \text{bar} & \text{bar} & \text{bar} \\
\end{array}
\]

\[
\text{SC(rating}='2')=1
\]

\[
V(\text{rating,R})=5
\]
Selections – Complex Predicates

- **Range Query:**
  - \( \text{sel}(A>a) = \frac{(A_{\text{max}} - a)}{(A_{\text{max}} - A_{\text{min}})} \)
  - Example: \( \text{sel}(\text{rating} \geq '2') \)

\[
\text{sel}(\text{rating} \geq '2') = \frac{(4 - 2)}{(4 - 0)} = \frac{1}{2}
\]

- **Negation Query**
  - \( \text{sel}(\text{not } P) = 1 - \text{sel}(P) \)
  - Example: \( \text{sel}(\text{rating} \neq '2') \)

\[
\text{sel}(\text{rating} \neq '2') = 1 - \text{sel}(\text{rating} = '2')
\]
• Negation Query
  – \( \text{sel(not } P) = 1 - \text{sel}(P) \)
  – Example: \( \text{sel(rating } \neq '2') \)

Observation: selectivity \( \approx \) probability

Conjunction:
  – \( \text{sel(rating = '2' AND name LIKE 'C%')} \)
  – \( \text{sel}(P_1 \land P_2) = \text{sel}(P_1) \cdot \text{sel}(P_2) \)
  – INDEPENDENCE ASSUMPTION

Disjunction:
  – \( \text{sel(rating = '2' OR name LIKE 'C%')} \)
  – \( \text{sel}(P_1 \lor P_2) \)
    \( = \text{sel}(P_1) + \text{sel}(P_2) - \text{sel}(P_1 \lor P_2) \)
    \( = \text{sel}(P_1) + \text{sel}(P_2) - \text{sel}(P_1) \cdot \text{sel}(P_2) \)
  – INDEPENDENCE ASSUMPTION, again
Selections – Complex Predicates

• Disjunction, in general:
  – \( \text{sel}(P_1 \text{ OR } P_2 \text{ OR } \ldots \text{ P}_n) = \)
  – \( 1 - (1 - \text{sel}(P_1)) \cdot (1 - \text{sel}(P_2)) \cdot \ldots \cdot (1 - \text{sel}(P_n)) \)

Joins

• Q: Given a join of R and S, what is the range of possible result sizes in #of tuples?

Result Size Estimation for Joins

• General case: \( R_{\text{cols}} \cap S_{\text{cols}} = \{ A \} \) where A is not a key for either table.
• Hint: for a given tuple of R, how many tuples of S will it match?
Cost Estimations

- Our formulas are nice but we assume that data values are uniformly distributed.

**Uniform Approximation of D**

- # of occurrences
- Distinct values of attribute

Cost Estimations

- Our formulas are nice but we assume that data values are uniformly distributed.

**Non-Uniform Approximation of D**

- Bucket #1 Count=8
- Bucket #2 Count=4
- Bucket #3 Count=15
- Bucket #4 Count=3
- Bucket #5 Count=14

Histograms with Quantiles

- Our formulas are nice but we assume that data values are uniformly distributed.

**Non-Uniform Approximation of D**

- A histogram type wherein the “spread” of each bucket is same.

**Equi-width Histogram ~ Quantiles**

- Bucket #1 Count=12
- Bucket #2 Count=12
- Bucket #3 Count=9
- Bucket #4 Count=12
Histograms with Quantiles

- A histogram type wherein the “spread” of each bucket is same.

![Equi-width Histogram - Quantiles](image)

Sampling

- Modern DBMSs also employ sampling to estimate predicate selectivities.

```
SELECT * FROM sailors
WHERE rating > 100
```

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Kayne</td>
<td>999</td>
<td>45.0</td>
</tr>
<tr>
<td>3</td>
<td>Obama</td>
<td>50</td>
<td>52.0</td>
</tr>
<tr>
<td>2</td>
<td>Tupac</td>
<td>32</td>
<td>26.0</td>
</tr>
<tr>
<td>6</td>
<td>Bieber</td>
<td>10</td>
<td>19.0</td>
</tr>
</tbody>
</table>

= 1/3

1 billion tuples

Today’s Class

- History & Background
- Relational Algebra Equivalences
- Plan Cost Estimation
- Plan Enumeration
Query Optimization

- Bring query in internal form into “canonical form” (syntactic q-opt)
- Generate alternative plans.
  - Single relation.
  - Multiple relations.
  - Nested sub-queries.
- Estimate cost for each plan.
- Pick the best one.

Plan Generation

- What are our plan options?

Sequential Scan

- **BR** (worst case)
- **BR/2** (on average, if we search for primary key)
Binary Search

- $\sim \log(B_R) + \frac{SC(A,R)}{F_R}$
- Extra blocks are ones that contain qualifying tuples

```
SELECT * FROM sailors
WHERE rating = 2
```

We showed that estimating this is non-trivial.

```
SELECT * FROM sailors
WHERE rating = 2
```

Index Search

- Index Search:
  - levels of index + blocks w/ qual. tuples

Case#1: Primary Key
Case#2: Secondary key – clustering index
Case#3: Secondary key – non-clust. index

```
SELECT * FROM sailors
WHERE rating = 2
```

```
SELECT * FROM sailors
WHERE rating = 2
```

Index Search: Case #1

- Primary Key
  - **cost:** $HT_i + 1$
Index Search: Case #2

- Secondary key with clustering index:
  - cost: $HT_i + \frac{SC(A,R)}{FR}$

Index Search: Case #3

- Secondary key with non-clustering index:
  - cost: $HT_i + SC(A,R)$

Query Optimization

- Bring query in internal form into “canonical form” (syntactic q-opt)
- Generate alternative plans.
  - Single relation.
  - Multiple relations.
    - Nested sub-queries.
- Estimate cost for each plan.
- Pick the best one.

Queries over Multiple Relations

- As number of joins increases, number of alternative plans grows rapidly
  - We need to restrict search space.
- **Fundamental decision in System R:** only left-deep join trees are considered.

Newer DBMSs don’t make this assumption anymore
Queries over Multiple Relations

- **Fundamental decision in System R:** only left-deep join trees are considered.

- Allows for fully pipelined plans where intermediate results not written to temp files.
- Not all left-deep trees are fully pipelined.

- **Enumerate the orderings**
  - *Example:* Left-deep tree #1, Left-deep tree #2...

- **Enumerate the plans for each operator**
  - *Example:* Hash, Sort-Merge, Nested Loop...

- **Enumerate the access paths for each table**
  - *Example:* Index #1, Index #2, Seq Scan...
Queries over Multiple Relations

- **Enumerate the orderings**
  - Example: Left-deep tree #1, Left-deep tree #2...
- **Enumerate the plans for each operator**
  - Example: Hash, Sort-Merge, Nested Loop...
- **Enumerate the access paths for each table**
  - Example: Index #1, Index #2, Seq Scan...
- Use **dynamic programming** to reduce the number of cost estimations.

Dynamic Programming Example

Compute the cheapest flight PIT -> PVG

*Solution: Compute partial optimal, left-to-right*
Compute the cheapest flight PIT -> PVG

**Solution: Compute partial optimal, left-to-right**
Compute the cheapest flight PIT -> PVG

Solution: Compute partial optimal, left-to-right
Q-Opt + Dynamic Programming

• Example: \( R \bowtie S \bowtie T \)

```
SELECT *
FROM R, S, T
WHERE R.a = S.a AND S.b = T.b
ORDER BY R.a
```
Q-Opt + Dynamic Programming

- $R \bowtie S \bowtie T$ order by $R.a$

Candidate Plan Example

- How to generate plans for search algorithm:
  1. Enumerate relation orderings
  2. Enumerate join algorithm choices
  3. Enumerate access method choices

```
SELECT sname, bname, day
FROM sailors S, reserves R, boats B
```

No real DBMSs does it this way. It’s actually more messy…
1. Enumerate relation orderings:

- \( S \Join R \Join B \)
- \( R \Join S \Join B \)
- \( B \Join S \Join \)
- \( B \Join R \Join S \)
- \( R \Join B \Join S \)

Prune plans with cross-products immediately!

2. Enumerate join algorithm choices:

- \( S \Join R \Join B \)
- \( R \Join S \Join B \)
- \( B \Join S \Join \)
- \( B \Join R \Join S \)
- \( R \Join B \Join S \)

Do this for the other plans.

3. Enumerate access method choices:

- \( S \Join R \Join B \)
- \( R \Join S \Join B \)
- \( B \Join S \Join \)
- \( B \Join R \Join S \)
- \( R \Join B \Join S \)

Do this for the other plans.
Postgres Optimizer

- Examines all types of join trees
  - Left-deep, Right-deep, bushy
- Two optimizer implementations:
  - Traditional Dynamic Programming Approach
  - Genetic Query Optimizer (GEQO)
- Postgres uses the traditional one when # of tables in query is less than 12 and switches to GEQO when there are 12 or more.
Query Optimization

- Bring query in internal form into “canonical form” (syntactic q-opt)
- Generate alternative plans.
  - Single relation.
  - Multiple relations.
- Nested sub-queries.
- Estimate cost for each plan.
- Pick the best one.

Nested Sub-Queries

- The DBMS treats nested sub-queries in the where clause as functions that take parameters and return a single value or set of values.
- Two Approaches:
  - Rewrite to de-correlate and/or flatten them
  - Decompose nested query and store result to temporary table
**Nested Sub-Queries: Rewrite**

```sql
SELECT name FROM sailors AS S
WHERE EXISTS ( 
    SELECT * FROM reserves AS R 
    WHERE S.sid = R.sid 
    AND R.day = '2016-10-24' 
) 
```

**Nested Sub-Queries: Decompose**

```sql
SELECT S.sid, MIN(R.day)
FROM sailors S, reserves R, boats B
WHERE S.sid = R.sid 
AND R.bid = B.bid 
AND B.color = 'red'
AND S.rating = (SELECT MAX(S2.rating) 
FROM sailors S2)
GROUP BY S.sid 
HAVING COUNT(*) > 1
```

*For each sailor with the highest rating (over all sailors) and at least two reservations for red boats, find the sailor id and the earliest date on which the sailor has a reservation for a red boat.*
Decomposing Queries

• For harder queries, the optimizer breaks up queries into blocks and then concentrates on one block at a time.
• Sub-queries are written to a temporary table that are discarded after the query finishes.

SELECT S.sid, MIN(R.day) 
FROM sailors S, reserves R, boats B 
WHERE S.sid = R.sid 
AND R.bid = B.bid 
AND B.color = 'red' 
AND S.rating = (SELECT MAX(S2.rating) 
                     FROM sailors S2) 
GROUP BY S.sid 
HAVING COUNT(*) > 1

SELECT MAX(rating) FROM sailors

### Nested Block

SELECT S.sid, MIN(R.day) 
FROM sailors S, reserves R, boats B 
WHERE S.sid = R.sid 
AND R.bid = B.bid 
AND B.color = 'red' 
AND S.rating = (SELECT MAX(S2.rating) 
                     FROM sailors S2) 
GROUP BY S.sid 
HAVING COUNT(*) > 1

### Nested Block
Decomposing Queries

SELECT MAX(rating) FROM sailors

SELECT S.sid, MIN(R.day)
FROM sailors S, reserves R, boats B
WHERE S.sid = R.sid
AND R.bid = B.bid
AND B.color = 'red'
AND S.rating = (SELECT MAX(S2.rating)
FROM sailors S2)
GROUP BY S.sid
HAVING COUNT(*) > 1

What Optimizers are Still Bad At

• Cardinality estimations are still hard.
• Problem Areas:
  – Prepared Statements
  – Correlated Columns
  – Plan Stability


Prepared Statements

SELECT S.name, B.bid
FROM sailors AS S,
reserves AS R, boats AS B
WHERE S.sid = R.sid
AND R.bid = B.bid
AND B.color = 'red'
AND S.rating > 1000

PREPARE myQuery AS
SELECT S.name, B.bid
FROM sailors AS S,
reserves AS R, boats AS B
WHERE S.sid = R.sid
AND R.bid = B.bid
AND B.color = 'red'
AND S.rating > 1000
EXECUTE myQuery;

PREPARE myQuery AS
SELECT S.name, B.bid
FROM sailors AS S,
     reserves AS R, boats AS B
WHERE S.sid = R.sid
AND R.bid = B.bid
AND B.color = 'red'
AND S.rating > 1000

EXECUTE myQuery('red', 1000);

• What should be the join order for SAILORS, BOATS, and RESERVES?

Prepared Statement Query Plan

\[\pi \text{name, bid} \] 
\[\sigma \text{rating>1000} \] 
\[\sigma \text{color='red'} \] 
\[\text{SAILORS} \] 
\[\text{BOAT} \] 
\[\text{RESERVES} \]
Prepared Statements

• **Solution #1** – Rerun optimizer each time the query is invoked.
• **Solution #2** – Generate multiple plans for different values of the parameters.
• **Solution #3** – Choose the average value for a parameter and use that for all invocations.

Correlated Columns

• We showed how selectivities are modeled as probabilities on whether a predicate on any given row will be satisfied.
• We then multiply these individual selectivities together.

Correlated Columns

• Consider a database of automobiles:
  – # of Makes = 10, # of Models = 100
• And the following query:
  – make="Honda" AND model="Accord"
• With the independence and uniformity assumption, the selectivity is:
  – 1/10 * 1/100 = 0.001
• But since only Honda makes Accords the real selectivity is 1/100 = 0.01.

Column Group Statistics

• Tell the DBMS that it should keep track of statistics for groups of columns together rather than just treating them all as independent variables.
• Only supported in commercial systems.
Plan Stability

- We want to deploy a new version of a DBMS but need to make sure that there are no performance regressions.
- What if 99% of the query plans are faster on the newer DBMS version, but 1% are slower?

Solution #1 – Allow tuning hints in plans.
Solution #2 – Set the optimizer version number and migrate queries one-by-one to the new optimizer.
Solution #3 – Save query plan from old version and provide it to the new DBMS.

Conclusions

- Ideas to remember:
  - Filter early as possible.
  - Selectivity estimations (uniformity, indep.; histograms; join selectivity)
  - Dynamic programming for join orderings
  - Rewrite nested queries
- Query optimization is hard…

Next Class

- How to refine database schemas to remove redundancies and prevent loss data.