Administrivia

• HW5 is due **Wed Oct 28**th.
• Mid-terms are available for **viewing**
  – Marilyn Walgara’s office (GHC 8120)
  – 9:00am-11:45am and 1:00pm-4:15pm
  – Bring your CMU id

Today’s Class

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

Cost-based Query Sub-System

Queries

Select * From Blah B
Where B.blah = blah

Query Parser

Query Optimizer

Plan Generator

Plan Cost Estimator

Catalog Manager

Schema

Statistics

Query Plan Evaluator
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.

IBM 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.

Predicate Pushdown

\[
\begin{align*}
\text{SELECT } & \text{cname, amt} \\
\text{FROM } & \text{customer, account} \\
\text{WHERE } & \text{customer.acctno = account.acctno} \\
& \text{AND account.amt > 1000}
\end{align*}
\]
Relational Algebra Equivalences

SELECT cname, amt
FROM customer, account
WHERE customer.acctno = account.acctno
AND account.amt > 1000

\[ \pi_{\text{cname, amt}}(\sigma_{\text{amt}>1000}(\text{customer} \bowtie \text{account})) = \pi_{\text{cname, amt}}(\text{customer} \bowtie (\sigma_{\text{amt}>1000}(\text{account}))) \]

Relational Algebra Equivalences

• Selections:
  - Perform them early
  - Break a complex predicate, and push
    \[ \sigma_{p_1 \land p_2 \land \ldots \land p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots(\sigma_{p_n}(R))\ldots)) \]
  - Simplify a complex predicate
    - \((X=Y \land Y=3) \rightarrow X=3 \land Y=3\)

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.)

Projection Pushdown

SELECT cname, amt
FROM customer, account
WHERE customer.acctno = account.acctno
AND account.amt > 1000

\[ \pi_{\text{cname, amt}}(\sigma_{\text{amt}>1000}(\text{customer})) \]

\[ \pi_{\text{acctno=acctno}}(\text{account}) \]
Relational Algebra Equivalences

- **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?

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 qualify?
- 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 ‘$A$’

Derivable Statistics

- $F_R \rightarrow$ max # records/block
- $B_R \rightarrow$ # blocks
- $SC(A,R) \rightarrow$ selection cardinality
  - avg # of records with $A$=given
  - $N_R / V(A,R)$
- Note that this assumes data uniformity
  - 10,000 students, 10 colleges – how many students in SCS?

Additional Statistics

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

- Where do we store them?
- How often do we update them?

- Manual invocations:
  - Postgres: \texttt{ANALYZE}
  - MySQL: \texttt{ANALYZE TABLE}

Selection Statistics

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

Selections – Complex Predicates

- Selectivity \texttt{sel}(P) of predicate \texttt{P}:
  \[
  \text{\texttt{sel}(P) = \frac{\text{SC}(P)}{N_R}}
  \]
- \# of tuples

Selections – Complex Predicates

- Assume that \texttt{V(rating, SAILORS)} has 5 distinct values (i.e., 0 to 4).
- simple predicate \texttt{P}: A=constant
  - \texttt{sel(A=constant) = \frac{1}{V(A,R)}}
  - eg., \texttt{sel(rating=“2”) = \frac{1}{5}}
- What if \texttt{V(A,R)} is unknown??
Selections – Complex Predicates

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

Selections – Complex Predicates

• Negation: \( \text{sel}(\text{rating} \neq '2') \)
  – \( \text{sel}(\text{not } P) = 1 - \text{sel}(P) \)
• Observation: selectivity \( \approx \) probability

Selections – Complex Predicates

• Conjunction:
  – \( \text{sel}(\text{rating} = '2' \text{ and } \text{name LIKE 'C%'}) \)
  – \( \text{sel}(P_1 \land P_2) = \text{sel}(P_1) \cdot \text{sel}(P_2) \)
  – INDEPENDENCE ASSUMPTION
  Not always true in practice!

Selections – Complex Predicates

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

- **Disjunction, in general:**
  - \( \text{sel}(P_1 \text{ or } P_2 \text{ or } \ldots \text{ or } P_n) = 1 - (1 - \text{sel}(P_1)) \cdot (1 - \text{sel}(P_2)) \cdot \ldots \cdot (1 - \text{sel}(P_n)) \)

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?

**Result Size Estimation for Joins**

- **General case:** \( R_{\text{cols}} \cap S_{\text{cols}} = \{A\} \) where \( A \) is not a key for either table.
  - Match each \( R \)-tuple with \( S \)-tuples:
    - \( \text{estSize} = \frac{N_R \cdot N_S}{V(A,S)} \)
  - Symmetrically, for \( S \):
    - \( \text{estSize} = \frac{N_R \cdot N_S}{V(A,R)} \)
  - **Overall:**
    - \( \text{estSize} = \frac{N_R \cdot N_S}{\max\{V(A,S), V(A,R)\}} \)
Cost Estimations

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

Histograms w/ Quantiles

- Allows the DBMS to have leverage better statistics about the data.

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?

Plan Generation

- Sequential Scan
- Binary Search
  - if sorted & consecutive
- Index Search
  - if an index exists

Sequential Scan

- $B_R$ (worst case)
- $B_R/2$ (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

### 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

We showed that estimating this is non-trivial.
Index Search: Case #2

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

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.
**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.

---

**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*
Q-Opt + Dynamic Programming

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

```
R ⨝ S ⨝ T
```

```
SELECT *
FROM R, S, T
WHERE R.a = S.a AND S.b = T.b
ORDER BY R.a
```

Candidate Plan Example

1. Enumerate relation orderings
2. Enumerate join algorithm choices
3. Enumerate access method choices
### Candidate Plans

**SELECT** `sname`, `bname`, `day`  
**FROM** `Sailors S`, `Reserves R`, `Boats B`  
**WHERE** `S.sid` = `R.sid` AND `R.bid` = `B.bid`

1. Enumerate relation orderings:

   - `R S B`
   - `S R B`
   - `B S R`
   - `S B R`
   - `B R S`
   - `R B S`

   **Prune plans with cross-products immediately!**

2. Enumerate join algorithm choices:

   - NLJ
   - HJ
   - Do this for the other plans.
Candidate Plans

3. Enumerate access method choices:

SELECT sname, bname, day
FROM Sailors S, Reserves R, Boats B

4. Now we can estimate the cost of each plan.

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 = '2015-10-18'
)
```

```sql
SELECT name
FROM Sailors AS S, Reserves AS R
WHERE S.sid = R.sid
AND R.day = '2015-10-18'
```

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.
What Optimizers are Still Bad At

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

More Information: Guy Lohman, "Query Optimization a "Solved" Problem?"

Prepared Statements

- What should be the join order for CUSTOMER and ACCOUNT?

Prepared Statement Query Plan

- 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.

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:
  - $1/10 \times 1/100 = 0.001$
- 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?
Plan Stability

• **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.

Query Optimizer Overview

• **System R:**
  – Break query in query blocks
  – Simple queries (i.e., no joins): look at stats
  – n-way joins: left-deep join trees; i.e., only one intermediate result at a time
    • **Pros:** smaller search space; pipelining
    • **Cons:** may miss optimal
  – 2-way joins: NL and sort-merge

Conclusions

• Ideas to remember:
  – Syntactic q-opt – do selections early
  – Selectivity estimations (uniformity, indep.; histograms; join selectivity)
  – Hash join (nested loops; sort-merge)
  – Left-deep joins
  – Dynamic programming

Next Class

• How to refine database schemas through normalization and functional dependencies to remove redundancies and prevent loss data.
Equivalence of Expressions

• Q: How to prove a transformation rule?
• Use relational tuple calculus to show that LHS = RHS:

\[ \sigma_p(R_1 \cup R_2) = \sigma_p(R_1) \cup \sigma_p(R_2) \]

LHS \hspace{2cm} RHS

QED

Equivalence of Expressions

\[ \sigma_p(R_1 \cup R_2) = \sigma_p(R_1) \cup \sigma_p(R_2) \]

\[ t \in LHS \iff t \in (R_1 \cup R_2) \land P(t) \iff (t \in R_1 \lor t \in R_2) \land P(t) \iff (t \in R_1 \land P(t)) \lor (t \in R_2 \land P(t)) \]

QED

Equivalence of Expressions

• Q: How to disprove a rule?

\[ \pi_A(R_1 - R_2) \neq (R_1 - \pi_A(R_2)) \]

R1

R2

\[ \begin{array}{c|c|c|c} A & B \\ \hline \text{Tramp} & \text{squirrels} \\ \text{Tramp} & \text{knifefights} \\ \end{array} \]