Query evaluation

Combining operators. Logical query optimization

By Marina Barsky
Winter 2016, University of Toronto
Quick recap: Relational Algebra Operators

**Core** operators:
- Selection $\sigma$
- Projection $\pi$
- Cartesian product $\times$
- Union $U$
- Difference $-$
- Renaming $\rho$

**Derived** operators:
- Join $\bowtie$
- Intersection $\cap$
Core RA operators
Slice operations: Projection

Producers from relation $\mathbf{R}$ a new relation that has only the $A_1, \ldots, A_n$ columns of $\mathbf{R}$.

$$S = \pi_{\text{attribute list}}(\mathbf{R})$$
Slice operations: Selection

Produces a new relation with those tuples of $R$ which satisfy condition $C$.

$S = \sigma_{\text{condition}}(R)$
Join operation: Cartesian product

1. Set of tuples \( rs \) that are formed by choosing the first part \((r)\) to be any tuple of \( R \) and the second part \((s)\) to be any tuple of \( S \).

2. Schema for the resulting relation is the union of schemas for \( R \) and \( S \).

3. If \( R \) and \( S \) happen to have some attributes in common, then prefix those attributes by the relation name.

\[ T = R \times S \]
Union

$T = R \cup S$
Difference

$R - S$
Renaming Operator

\[ \rho_{S(A_1,A_2,\ldots,A_n)} (R) \]

1. Resulting relation has exactly the same tuples as \( R \), but the name of the relation is \( S \).

2. Moreover, the attributes of the result relation \( S \) are named \( A_1, A_2, \ldots, A_n \), in order from the left.
Query with renaming: example

T (uid1, uid2)

A → B
B → A
B → C
A → C
C → B

• Find all true friends in twitter dataset

• By renaming T we created two identical relations R and S, and we now extract all tuples where for each pair X → Y in R there is a pair Y → X in S

```
SELECT R.uid1, R.uid2
FROM T as R, T as S
WHERE R.uid1 = S.uid2
AND R.uid2 = S.uid1
```

\[ \pi_{R.uid1, R.uid2} \sigma_{R.uid1=S.uid2 \text{ AND } R.uid2=S.uid1}(\rho_R(T) \times \rho_S(T)) \]
Core operators – sufficient to express any query in relational model

• Relational model due to Edgar “Ted” Codd, a mathematician at IBM in 1970
  • A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* 13 (6): 377–387

• He proved that any query can be expressed using these core operators: $\sigma, \pi, x, U, −, \rho$

The Relational model is **precise, implementable**, and we can operate on it (query/update, etc.)
Note that any RA Operator returns relation, so we can compose complex queries from known operators

\[
\pi_{\text{sname, gpa}}(\sigma_{\text{gpa} > 3.5}(\text{Students}))
\]

\[
\sigma_{\text{gpa} > 3.5}(\pi_{\text{sname, gpa}}(\text{Students}))
\]

Are these logically equivalent?
RA has Limitations!

- Cannot compute “transitive closure”

<table>
<thead>
<tr>
<th>Name1</th>
<th>Name2</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Mary</td>
<td>Father</td>
</tr>
<tr>
<td>Mary</td>
<td>Joe</td>
<td>Cousin</td>
</tr>
<tr>
<td>Mary</td>
<td>Bill</td>
<td>Spouse</td>
</tr>
<tr>
<td>Nancy</td>
<td>Lou</td>
<td>Sister</td>
</tr>
</tbody>
</table>

- Find all direct and indirect relatives of Fred
- Cannot express in RA!!!
  - Need to write C program, use a graph engine, or PL-SQL...
Derived RA operators
Join operation: Theta-join

1. The result of this operation is constructed as follows:
   
   a) Take the Cartesian product of \( R \) and \( S \).
   b) Select from the product only those tuples that satisfy the condition \( C \).

2. Schema for the result is the union of the schema of \( R \) and \( S \), with “\( R \)” or “\( S \)” prefix as necessary.

\[ T = R \bowtie_{\text{condition}} S \]

Shortcut for
\[ T = \sigma_{\text{condition}} (R \times S) \]
Join operation: Equijoin

1. Equijoin is a subset of theta-joins where the join condition is equality.

$T = \sigma_{R.A = S.B} (R \times S)$

Shortcut for

$T = \sigma_{R.A = S.B} (R \times S)$
Natural Join

Special case of equijoin when attributes we want to use in join have the same name in both tables

\[ R \bowtie S \]

Let \( A_1, A_2, \ldots, A_n \) be the attributes in both the schema of \( R \) and the schema of \( S \).

Then a tuple \( r \) from \( R \) and a tuple \( s \) from \( S \) are successfully paired if and only if \( r \) and \( s \) agree on each of the attributes \( A_1, A_2, \ldots, A_n \).
Outer join

1. For each tuple in R, include all tuples in S which satisfy join condition, but include also tuples of R that do not have matches in S

2. For this case, pair tuples of R with NULL

Left outer join

\[ T = R \bowtie_{\text{condition}} S \]
Outer join: example

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>99999</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>44444</td>
<td>flue</td>
</tr>
<tr>
<td>33</td>
<td>66666</td>
<td>lung</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>99999</td>
<td>lawyer</td>
</tr>
<tr>
<td>20</td>
<td>44444</td>
<td>cashier</td>
</tr>
</tbody>
</table>

Anonymous patient P

Anonymous occupation O

\[
T = P \bowtie O
\]

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>job</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>99999</td>
<td>heart</td>
<td>lawyer</td>
</tr>
<tr>
<td>20</td>
<td>44444</td>
<td>flue</td>
<td>cashier</td>
</tr>
<tr>
<td>33</td>
<td>66666</td>
<td>lung</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Quick question

If I have a relation R with 100 records and a relation S with exactly 1 record, how many records will be in the result of R LEFT OUTER JOIN S?

A. At least 100, but could be more
B. Could be any number between 0 and 100 inclusive
C. 0
D. 1
E. 100
Quick question

If I have a relation R with 100 records and a relation S with exactly 1 record, how many records will be in the result of R LEFT OUTER JOIN S?

A. At least 100, but could be more
B. Could be any number between 0 and 100 inclusive
C. 0
D. 1
E. 100
Intersection

\[ T = R \cap S \]
Why intersection is not a “core” operation?

R \cap S

R - S (are in R but not in S)

R - (R-S)
Why intersection is not a “core” operation?

$R \cap S$ shortcut to

$R - (R - S)$

Can be derived using core operations
Set vs. bag (multi-set) semantics

- **Sets**: \{a,b,c\}, \{a,d,e,f\}, ...
- **Bags**: \{a,a,b,c\}, \{b,b,b,b,b\}, ...

- Relational algebra has two semantics:
  - Set semantics = standard relational algebra
  - Bag semantics = extended Relational Algebra

- Rule of thumb:
  - Every paper will assume set semantics
  - Every implementation will assume bag semantics
Operations on multisets

All RA operations need to be defined carefully on bags

• $\sigma_C(R)$: preserve the number of occurrences

• $\pi_A(R)$: no duplicate elimination

• Cross-product, join: no duplicate elimination

This is important- relational DBMSs work on multisets, not sets!
Quick question

What implementation would have a smaller cost – implementation for bag projection or set projection? Why?

A. Set projection. The number of records in a set is typically smaller than a bag, and cost is a function of the number of records in the collection.

B. Bag projection. A bag is easier to reason about formally, and therefore allows more aggressive optimization opportunities.

C. Bag projection. Removing duplicates requires an extra step, which can be expensive and is not always required by the application.
Quick question

What implementation would have a smaller cost – implementation for bag projection or set projection? Why?

A. Set projection. The number of records in a set is typically smaller than a bag, and cost is a function of the number of records in the collection.

B. Bag projection. A bag is easier to reason about formally, and therefore allows more aggressive optimization opportunities.

C. Bag projection. Removing duplicates requires an extra step, which can be expensive and is not always required by the application.
Extended operators on bags

• Duplicate elimination $\delta$
• Sorting $\tau$
• Grouping and aggregation $\gamma$
RDBMS query evaluation

How does a RDBMS answer your query?

- **SQL Query**: Declarative query (user declares what results are needed)
- **Relational Algebra (RA) Plan**: Translate to relational algebra expression
- **Optimized RA Plan**: Find logically equivalent—but more efficient—RA expression
- **Execution**: Execute each operator of the optimized plan!
RDBMS query optimizer: steps

- Convert parsed SQL into corresponding RA expression
- Apply known algebraic transformations – produce improved logical query plan
- Transform based on estimated cost
- Choose one min-cost logical expression
- For each step, consider alternative physical implementations
- Choose physical plan with min I/Os
- Execute

**question**

SQL query

**parse**

parse tree

**convert**

logical query plan

**improve logically**

“improved” l.q.p

**estimate sizes**

l.q.p. +sizes

consider physical plans

{P1, P2, .....}

**estimate costs**

pick best

execute

**answer**
RDBMS query evaluation

How does a RDBMS answer your query?

1. **SQL Query**
   - Declarative query (user declares what results are needed)

2. **Relational Algebra (RA) Plan**
   - Translate to relational algebra expression

3. **Optimized RA Plan**
   - Find logically equivalent but more efficient RA expression

4. **Execution**
   - Execute each operator of the optimized plan!

*That we just learned how to do!*
Example: two SQL queries – the same execution plan

\[ R \bowtie_{R.A = S.B} S \quad \leftrightarrow \quad (R \times S) \]  

\[ \sigma_{R.A = S.B} (R \times S) \]

Join

Cross-product with selection

- `SELECT * FROM R JOIN S ON R.A = S.B`
- `SELECT * FROM R, S WHERE R.A = S.B`

- Two ways to request the same results
- The optimizer does not care about the syntax of SQL query: it is going to work on the algebraic representation anyway
- Because the algebraic expressions are equivalent, the optimizer will have the same final plan for both queries
RDBMS query evaluation

How does a RDBMS answer your query?

Relational Algebra allows us to translate declarative (SQL) queries into precise and optimizable expressions!
RDBMS query optimizer: steps

- Convert parsed SQL into corresponding RA expression
- Apply known algebraic transformations – produce improved logical query plan
- Transform based on estimated cost
- Choose one min-cost logical expression
- For each step, consider alternative physical implementations
- Choose physical plan with min I/Os
- Execute
Translating general SQL queries (SELECT-FROM-WHERE) into RA

• What is the general form of an SFW query in RA?

• Given a general SFW SQL query:

```
SELECT A_1, ..., A_n
FROM R_1, ..., R_m
WHERE c_1, ..., c_k;
```

• We can express this in relational algebra as follows:

```
\pi_{A_1, \ldots, A_n}(\sigma_{c_1} \cdots \sigma_{c_k}(R_1 \times \ldots \times R_m))
```
We can visualize the RA expression as a tree

$$\pi_B(R(A,B) \bowtie S(B,C))$$

Bottom-up tree traversal = order of operation execution!
From RA to SQL

What SQL query does this correspond to?

Are there any logically equivalent RA expressions?
From SQL to RA: example 1

\[ \pi_{A,D}(\sigma_{A<10}(T \bowtie (R \bowtie S))) \]
From SQL to RA: example 2

\[ S \ (\text{product}, \text{city}, \text{price}) \]

\[
\begin{align*}
\text{SELECT} & \quad \text{city, count (*)} \\
\text{FROM} & \quad S \\
\text{GROUP BY} & \quad \text{city} \\
\text{HAVING} & \quad \text{sum(price)} > 100
\end{align*}
\]
RDBMS query evaluation

How does a RDBMS answer your query?

SQL Query → Relational Algebra (RA) Plan → *Optimized RA Plan* → Execution

We transform the original RA expression into equivalent expressions using algebraic laws
RDBMS query optimizer: steps

- Convert parsed SQL into corresponding RA expression
- Apply known algebraic transformations – produce improved logical query plan
- Transform based on estimated cost
- Choose one min-cost logical expression
- For each step, consider alternative physical implementations
- Choose physical plan with min I/Os
- Execute

**question**

1. Parse SQL query
2. Parse tree
3. Convert
4. Logical query plan
5. Improve logically
6. “Improved” l.q.p
7. Estimate sizes
8. l.q.p. + sizes
9. Consider physical plans
10. \{P1,P2,.....\}
11. Estimate costs
12. Pick best
13. Execute
14. Answer
RA laws involving selection (\(\sigma\))

Selection for a single relation

- Splitting law:
  \[ \sigma_{C \wedge D}(R) = \sigma_C(\sigma_D(R)) \]

- Commutative law: order is flexible
  \[ \sigma_C(\sigma_D(R)) = \sigma_D(\sigma_C(R)) \]
RA laws involving selection ($\sigma$)

**Binary** selection (on 2 relations)

\[
\sigma_C(R \times S) = R \triangleright \triangleleft C \ S
\]

\[
\sigma_C(R \triangleright \triangleleft S) = \sigma_C(R) \triangleright \triangleleft S
\]

\[
\sigma_C(R \triangleright \triangleleft D S) = \sigma_C(R) \triangleright \triangleleft D S
\]

For the binary operators, we **push the selection to** $R$ only if all attributes in the condition $C$ are in $R$. 
Pushing selections: example

Consider \( R(A,B) \) and \( S(B,C) \) and the expression below:

\[
\sigma_{A=1} \land B < C (R \bowtie S)
\]

1. Splitting AND

\[
\sigma_{A=1} (\sigma_{B < C} (R \bowtie S))
\]

2. Push \( \sigma \) to \( S \)

\[
\sigma_{A=1} (R \bowtie \sigma_{B < C} (S))
\]

3. Push \( \sigma \) to \( R \)

\[
\sigma_{A=1} (R \bowtie \sigma_{B < C} (S))
\]
Laws for (bag) projection

**A simple law:** Project out attributes that are not needed later.

- i.e. keep only the output attr. and any join attribute.

\[
\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))
\]

\[
\pi_L(R \bowtie_C S) = \pi_L(\pi_M(R) \bowtie_C \pi_N(S))
\]

\[
\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))
\]

\[
\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))
\]
Pushing projection: example

Schema $R(a,b,c), S(c,d,e)$

$$\pi_{a+e\to x}(R \bowtie S) \equiv \pi_{a+e\to x}(\pi_{a,c}(R) \bowtie \pi_{c,e}(S))$$

$$\pi_{a+b\to x,d+e\to y}(R \bowtie S) \equiv \pi_{x,y}(\pi_{a+b\to x,c}(R) \bowtie \pi_{d+e\to y,c}(S))$$
Why to push projections?

Why might we prefer this plan?
Commutative and associative laws for joins

- Commutative and associative laws for joins:

\[ R \bowtie S = S \bowtie R \]
\[ (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \]

Above laws are applicable for both sets and bags
Quick question

• Given relation R(A,B):

  • Here, projection & selection commute:
    • $\sigma_{A=5}(\pi_{A}(R)) \leftrightarrow \pi_{A}(\sigma_{A=5}(R))$

  • What about here?
    • $\pi_{B}(\sigma_{A=5}(R)) \leftrightarrow \sigma_{A=5}(\pi_{B}(R))$?
Logical optimization: example

\[ \pi_{A,D}(\sigma_{A<10}(T \bowtie (R \bowtie S))) \]

Push down selection on A so it occurs earlier
Logical optimization: example

\[
\pi_{A,D}(T \bowtie (\sigma_{A<10}(R) \bowtie S))
\]

SELECT R.A, T.D
FROM R, S, T
WHERE R.B = S.B
AND S.C = T.C
AND R.A < 10;

Push down selection on A so it occurs earlier
Logical optimization: example

\[ \pi_{A,D}(T \bowtie (\sigma_{A<10}(R) \bowtie S)) \]

\[ \text{SELECT } R.A, T.D \]
\[ \text{FROM } R, S, T \]
\[ \text{WHERE } R.B = S.B \]
\[ \text{AND } S.C = T.C \]
\[ \text{AND } R.A < 10; \]
Logical optimization: example

\[
\pi_{A,D} \left( T \bowtie \pi_{A,C} \left( \sigma_{A<10}(R) \bowtie S \right) \right)
\]

We eliminate B earlier!

In general, when is an attribute not needed...?
Improving logical query plans using algebraic laws: summary

1. Push $\sigma$ as far down as possible

2. Do splitting of complex conditions in $\sigma$ in order to push $\sigma$ even further

3. Push $\pi$ as far down as possible, introduce new early $\pi$ (but take care for exceptions)

4. Combine $\sigma$ with $\times$ to produce $\Theta$-joins or equijoins

5. Choose an order for joins

Topic by itself
Why still so many different plans selected for the same query? Depends on sizes of intermediate outputs

SELECT extendedprice
FROM lineitem, supplier
WHERE lineitem.sID = supplier.sID
AND extendedprice: varies
AND supplier.accountbalance: varies

- Same query executed with different selection cardinality – covering from 0 to 100% of all values – results in completely different plans
- Here: 89 plans, each in different color

Picasso Database Query Optimizer Visualizer: [link]
Coming next: cost-based transformations