

Query Evaluation and Optimization

Jan Chomicki
University at Buffalo

Evaluating $\sigma_E(R)$

Evaluating $\sigma_E(R)$

Pick the option of least estimated cost.

Evaluating $\sigma_E(R)$

Pick the option of least estimated cost.

Any E

Linear (sequential) scan.

Evaluating $\sigma_E(R)$

Pick the option of least estimated cost.

Any E

Linear (sequential) scan.

E is $A = c$

- if file sorted on A , use binary search
- if file has an index on A , use the index
- if file hashed on A , use hashing

Evaluating $\sigma_E(R)$

Pick the option of least estimated cost.

Any E

Linear (sequential) scan.

E is $A = c$

- if file sorted on A , use binary search
- if file has an index on A , use the index
- if file hashed on A , use hashing

E is $A\theta c$ for $\theta \in \{<, \leq, >, \geq\}$

- if file sorted on A , then:
 - ▶ use binary search
 - ▶ if there is an index on A , use the index

Complex selection conditions

Complex selection conditions

E is $E_1 \wedge E_2$

- use E_1 or E_2 to narrow down the search
- use composite index (if available)
- use indexes for E_1 and E_2 to obtain sets of RowIds, then intersect those sets
- use bitmap indexes for E_1 and E_2 to obtain bitvectors, then use boolean AND
- use a combination of bitmap and other kinds of indexes

Complex selection conditions

E is $E_1 \wedge E_2$

- use E_1 or E_2 to narrow down the search
- use composite index (if available)
- use indexes for E_1 and E_2 to obtain sets of RowIds, then intersect those sets
- use bitmap indexes for E_1 and E_2 to obtain bitvectors, then use boolean AND
- use a combination of bitmap and other kinds of indexes

E is $E_1 \vee E_2$

- E_1 or E_2 cannot be used separately to narrow down the search
- use indexes for E_1 and E_2 to obtain sets of RowIds, then take the union of those sets
- use bitmap indexes for E_1 and E_2 to obtain bitvectors, then use boolean OR
- use a combination of bitmap and other kinds of indexes

Evaluating $R \bowtie_{A=B} S$

Evaluating $R \bowtie_{A=B} S$

Nested loops

```
foreach  $t \in R$  do
  foreach  $s \in S$  do
    if  $t[A] = s[B]$  then output  $(t, s)$ 
```

Evaluating $R \bowtie_{A=B} S$

Nested loops

```
foreach  $t \in R$  do
  foreach  $s \in S$  do
    if  $t[A] = s[B]$  then output  $(t, s)$ 
```

Index join (S has an index on B)

```
foreach  $t \in R$  do
  lookup  $S$  using the index
  and the value  $t[A]$ 
```

Evaluating $R \bowtie_{A=B} S$

Nested loops

```
foreach  $t \in R$  do
  foreach  $s \in S$  do
    if  $t[A] = s[B]$  then output  $(t, s)$ 
```

Index join (S has an index on B)

```
foreach  $t \in R$  do
  lookup  $S$  using the index
  and the value  $t[A]$ 
```

Sort-merge join (R sorted on A , S sorted on B)

```
scan files in sorted order
matching  $A$ -values in  $R$ 
with  $B$ -values in  $S$ 
```


Hash-join

```
create new hashed files  $HR$  and  $HS$ ;  
foreach  $t \in R$  do  
   $i := h(t[A])$ ;  
  store  $t$  in bucket  $r_i$  of  $HR$ ;  
foreach  $t \in S$  do  
   $i := h(t[A])$ ;  
  store  $t$  in bucket  $s_i$  of  $HS$ ;  
for each bucket  $r_i$  of  $HR$   
  read  $r_i$  and build an in-memory hash index  
  for each tuple  $t$  in the bucket  $s_i$  of  $S$   
    probe the hash index to find and output  
    the tuples  $w$  such that  $w[A] = t[A]$ 
```

Hash-join

```
create new hashed files  $HR$  and  $HS$ ;  
foreach  $t \in R$  do  
   $i := h(t[A])$ ;  
  store  $t$  in bucket  $r_i$  of  $HR$ ;  
foreach  $t \in S$  do  
   $i := h(t[A])$ ;  
  store  $t$  in bucket  $s_i$  of  $HS$ ;  
for each bucket  $r_i$  of  $HR$   
  read  $r_i$  and build an in-memory hash index  
  for each tuple  $t$  in the bucket  $s_i$  of  $S$   
    probe the hash index to find and output  
    the tuples  $w$  such that  $w[A] = t[A]$ 
```

- the hash function used to build the hash index and to probe it has to be different than h
- each bucket of HR has to fit in main memory

Oracle: selection and join

Oracle: selection and join

Equality condition $A = c$

- unique index on A : INDEX UNIQUE SCAN
- nonunique index on A : INDEX RANGE SCAN

Oracle: selection and join

Equality condition $A = c$

- unique index on A : INDEX UNIQUE SCAN
- nonunique index on A : INDEX RANGE SCAN

Range condition on A

- any index on A : INDEX RANGE SCAN

Oracle: selection and join

Equality condition $A = c$

- unique index on A : INDEX UNIQUE SCAN
- nonunique index on A : INDEX RANGE SCAN

Range condition on A

- any index on A : INDEX RANGE SCAN

Conjunction

- both indexes: AND-EQUAL (intersection of Rowid sets)

Oracle: selection and join

Equality condition $A = c$

- unique index on A : INDEX UNIQUE SCAN
- nonunique index on A : INDEX RANGE SCAN

Range condition on A

- any index on A : INDEX RANGE SCAN

Conjunction

- both indexes: AND-EQUAL (intersection of Rowid sets)

Join

- 1 SORT JOIN/MERGE JOIN (sort-merge)
- 2 NESTED LOOPS (nested loop or index join)
- 3 HASH JOIN

Query optimization

Query optimization

Stages:

- 1 Parsing, validation, view expansion.
- 2 Logical plan selection (using algebraic laws).
- 3 Physical plan selection (cost-based)

Algebraic laws

Algebraic laws

Assumption

Tuples are mappings from attribute names to domain values.

Algebraic laws

Assumption

Tuples are mappings from attribute names to domain values.

Join (and Cartesian product)

$$E_1 \bowtie E_2 = E_2 \bowtie E_1$$

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3).$$

Algebraic laws

Assumption

Tuples are mappings from attribute names to domain values.

Join (and Cartesian product)

$$E_1 \bowtie E_2 = E_2 \bowtie E_1$$

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3).$$

Cascading

$$\pi_{A_1, \dots, A_n}(\pi_{A_1, \dots, A_n, \dots, A_{n+k}}(E)) = \pi_{A_1, \dots, A_n}(E)$$

$$\sigma_{F_1}(\sigma_{F_2}(E)) = \sigma_{F_1 \wedge F_2}(E)$$

Commuting projections

Commuting projections

With selection

$$\pi_{A_1, \dots, A_n}(\sigma_F(E)) = \sigma_F(\pi_{A_1, \dots, A_n}(E))$$

if F does not involve any attributes other than A_1, \dots, A_n .

Commuting projections

With selection

$$\pi_{A_1, \dots, A_n}(\sigma_F(E)) = \sigma_F(\pi_{A_1, \dots, A_n}(E))$$

if F does not involve any attributes other than A_1, \dots, A_n .

With Cartesian product

$$\pi_{A_1, \dots, A_n, A_{n+1}, \dots, A_{n+k}}(E_1 \times E_2) =$$
$$\pi_{A_1, \dots, A_n}(E_1) \times \pi_{A_{n+1}, \dots, A_{n+k}}(E_2)$$

where A_1, \dots, A_n come from E_1 and A_{n+1}, \dots, A_{n+k} come from E_2 .

Commuting projections

With selection

$$\pi_{A_1, \dots, A_n}(\sigma_F(E)) = \sigma_F(\pi_{A_1, \dots, A_n}(E))$$

if F does not involve any attributes other than A_1, \dots, A_n .

With Cartesian product

$$\pi_{A_1, \dots, A_n, A_{n+1}, \dots, A_{n+k}}(E_1 \times E_2) =$$
$$\pi_{A_1, \dots, A_n}(E_1) \times \pi_{A_{n+1}, \dots, A_{n+k}}(E_2)$$

where A_1, \dots, A_n come from E_1 and A_{n+1}, \dots, A_{n+k} come from E_2 .

With union

$$\pi_{A_1, \dots, A_n}(E_1 \cup E_2) = \pi_{A_1, \dots, A_n}(E_1) \cup \pi_{A_1, \dots, A_n}(E_2).$$

Commuting selections

Commuting selections

With Cartesian product

$$\sigma_F(E_1 \times E_2) = \sigma_F(E_1) \times E_2$$

if F involves only the attributes of E_1 .

Commuting selections

With Cartesian product

$$\sigma_F(E_1 \times E_2) = \sigma_F(E_1) \times E_2$$

if F involves only the attributes of E_1 .

With union

$$\sigma_F(E_1 \cup E_2) = \sigma_F(E_1) \cup \sigma_F(E_2).$$

Commuting selections

With Cartesian product

$$\sigma_F(E_1 \times E_2) = \sigma_F(E_1) \times E_2$$

if F involves only the attributes of E_1 .

With union

$$\sigma_F(E_1 \cup E_2) = \sigma_F(E_1) \cup \sigma_F(E_2).$$

With set difference

$$\sigma_F(E_1 - E_2) = \sigma_F(E_1) - \sigma_F(E_2).$$

Cost-based query optimization

Logical query plan

- expression in relational algebra
- produced by rewrite rules that are obtained from the algebraic laws.

Physical query plan

- *operators*: different implementations of relational algebra operators
- *table-scan, index-scan, sort-scan, sort.*

Cost-based query optimization

Logical query plan

- expression in relational algebra
- produced by rewrite rules that are obtained from the algebraic laws.

Physical query plan

- *operators*: different implementations of relational algebra operators
- *table-scan, index-scan, sort-scan, sort.*

A **physical** query plan of **least estimated cost** is produced from a **logical** query plan.

Choices to be made

Choices to be made

- 1 **order** and **grouping** of joins
- 2 choosing an **algorithm** for each operator occurrence
- 3 adding other operators like **sorting**
- 4 materialization vs. pipelining

Choices to be made

- 1 **order** and **grouping** of joins
- 2 choosing an **algorithm** for each operator occurrence
- 3 adding other operators like **sorting**
- 4 materialization vs. pipelining

Assumptions

- relation \equiv file
- cost \equiv size of intermediate relations

Cost estimation using size estimates

Cost estimation using size estimates

Size estimates

- accurate
- easy to compute
- consistent

Cost estimation using size estimates

Size estimates

- accurate
- easy to compute
- consistent

Notation

- $B(R)$: number of blocks in relation R
- $T(R)$: number of tuples in relation R
- $V(R, A)$: number of distinct values R has in attribute A (can be generalized to *sets* of attributes)

Relational algebra operators

Relational algebra operators

Projection

$$T(\pi_X(R)) = T(R)$$

Relational algebra operators

Projection

$$T(\pi_X(R)) = T(R)$$

Selection

- $T(\sigma_{A=c}(R)) = T(R)/V(R, A)$
- $T(\sigma_{A < c}(R)) = T(R)/3$
- $T(\sigma_{C_1 \wedge C_2}(R)) = T(\sigma_{C_1}(\sigma_{C_2}(R)))$
- $T(\sigma_{C_1 \vee C_2}(R)) = \min(T(R), T(\sigma_{C_1}(R)) + T(\sigma_{C_2}(R)))$

Relational algebra operators

Projection

$$T(\pi_X(R)) = T(R)$$

Selection

- $T(\sigma_{A=c}(R)) = T(R)/V(R, A)$
- $T(\sigma_{A<c}(R)) = T(R)/3$
- $T(\sigma_{C_1 \wedge C_2}(R)) = T(\sigma_{C_1}(\sigma_{C_2}(R)))$
- $T(\sigma_{C_1 \vee C_2}(R)) = \min(T(R), T(\sigma_{C_1}(R)) + T(\sigma_{C_2}(R)))$

Those estimates may be **inconsistent**.

Natural join $R(X, Y) \bowtie S(Y, Z)$

Simplifying assumptions

- **containment:** $V(R, Y) \leq V(S, Y)$ implies $\pi_Y(R) \subseteq \pi_Y(S)$
- **preservation:** $V(R \bowtie S, A) = V(R, A)$.

Natural join $R(X, Y) \bowtie S(Y, Z)$

Simplifying assumptions

- **containment:** $V(R, Y) \leq V(S, Y)$ implies $\pi_Y(R) \subseteq \pi_Y(S)$
- **preservation:** $V(R \bowtie S, A) = V(R, A)$.

Y consists of one attribute A

$$T(R \bowtie S) = \frac{T(R)T(S)}{\max(V(R, A), V(S, A))}.$$

Natural join $R(X, Y) \bowtie S(Y, Z)$

Simplifying assumptions

- **containment:** $V(R, Y) \leq V(S, Y)$ implies $\pi_Y(R) \subseteq \pi_Y(S)$
- **preservation:** $V(R \bowtie S, A) = V(R, A)$.

Y consists of one attribute A

$$T(R \bowtie S) = \frac{T(R)T(S)}{\max(V(R, A), V(S, A))}.$$

Y consists of n attributes A_1, \dots, A_n

$$T(R \bowtie S) = \frac{T(R)T(S)}{\prod_{j=1, \dots, n} \max(V(R, A_j), V(S, A_j))}.$$

Multiway join $S = R_1 \bowtie \dots \bowtie R_k$

Assumption

A_1, \dots, A_n are the attributes that appear in more than one relation.

Multiway join $S = R_1 \bowtie \dots \bowtie R_k$

Assumption

A_1, \dots, A_n are the attributes that appear in more than one relation.

A -factor $M(A)$

Product of $V(R_i, A)$ for every relation R_i in which A appears, except for the smallest $V(R_i, A)$.

Multiway join $S = R_1 \bowtie \dots \bowtie R_k$

Assumption

A_1, \dots, A_n are the attributes that appear in more than one relation.

A-factor $M(A)$

Product of $V(R_i, A)$ for every relation R_i in which A appears, except for the smallest $V(R_i, A)$.

Multiway join size

$$T(R_1 \bowtie \dots \bowtie R_k) = \frac{\prod_{i=1, \dots, k} T(R_i)}{\prod_{j=1, \dots, n} M(A_j)}.$$

Properties

- **consistency**: join size estimate is independent of the order and the grouping of the joins
- **accuracy**: better estimates exist, for example based on **histograms**

Properties

- **consistency**: join size estimate is independent of the order and the grouping of the joins
- **accuracy**: better estimates exist, for example based on **histograms**

Statistics

- computed periodically, on-demand or incrementally
- do not have to be very accurate

Computing the best plan for $R_1 \bowtie \dots \bowtie R_k$

Computing the best plan for $R_1 \bowtie \dots \bowtie R_k$

Constructing a table $D(\text{Size}, \text{LeastCost}, \text{BestPlan})$

- Base cases:
 - ▶ One relation R : $D(T(R), 0, R)$
 - ▶ One join $R_i \bowtie R_j$: $D(T(R_i \bowtie R_j), 0, R_i \bowtie R_j)$ if $T(R_i) \leq T(R_j)$
- Inductive case $\bowtie_{\mathcal{R}}$:
 - 1 $\mathcal{R} = \{R_{i_1}, \dots, R_{i_m}\}$ for $m < k$
 - 2 for every $j \in \{i_1, \dots, i_m\}$, retrieve the tuple $D(T(\bigwedge_{i \neq j, R_i \in \mathcal{R}} R_i), C_j, E_j)$
 - 3 find $p \in \{i_1, \dots, i_m\}$ such that $C_p + T(\bigwedge_{i \neq p, R_i \in \mathcal{R}} R_i)$ is minimal
 - 4 let $Best = (\bigwedge_{i \neq p, R_i \in \mathcal{R}} R_i)$
 - 5 return $D(T(Best \bowtie R_p), C_p + T(Best), E_p \bowtie R_p)$

Computing the best plan for $R_1 \bowtie \dots \bowtie R_k$

Constructing a table $D(\text{Size}, \text{LeastCost}, \text{BestPlan})$

- Base cases:
 - ▶ One relation R : $D(T(R), 0, R)$
 - ▶ One join $R_i \bowtie R_j$: $D(T(R_i \bowtie R_j), 0, R_i \bowtie R_j)$ if $T(R_i) \leq T(R_j)$
- Inductive case $\bowtie_{\mathcal{R}}$:
 - 1 $\mathcal{R} = \{R_{i_1}, \dots, R_{i_m}\}$ for $m < k$
 - 2 for every $j \in \{i_1, \dots, i_m\}$, retrieve the tuple $D(T(\bigwedge_{i \neq j, R_i \in \mathcal{R}} R_i), C_j, E_j)$
 - 3 find $p \in \{i_1, \dots, i_m\}$ such that $C_p + T(\bigwedge_{i \neq p, R_i \in \mathcal{R}} R_i)$ is minimal
 - 4 let $Best = (\bigwedge_{i \neq p, R_i \in \mathcal{R}} R_i)$
 - 5 return $D(T(Best \bowtie R_p), C_p + T(Best), E_p \bowtie R_p)$

How many plans are considered?

Refinements

More sophisticated cost measure

- calculating the cost of using different algorithms for the same operation
- keeping multiple plans, together with their costs, to account for *interesting* join orders.

Refinements

More sophisticated cost measure

- calculating the cost of using different algorithms for the same operation
- keeping multiple plans, together with their costs, to account for *interesting* join orders.

Greedy algorithm

- 1 start with a pair of relations whose estimated join size is minimal
- 2 keep adding further relations with minimal estimated join size

Completing the evaluation plan

Operations

- TableScan(Relation)
- SortScan(Relation, Attributes)
- IndexScan(Relation, Condition)
- Filter(Condition)
- Sort(Attributes)
- different algorithms for the basic operations

Completing the evaluation plan

Operations

- TableScan(Relation)
- SortScan(Relation, Attributes)
- IndexScan(Relation, Condition)
- Filter(Condition)
- Sort(Attributes)
- different algorithms for the basic operations

Pipelining vs. materialization.

