Relational operations

• We will consider how to implement and define physical operators for:
  – Projection
  – Selection
  – Grouping
  – Set operators
  – Join

• Then we will discuss how the optimizer use them to generate physical query plans
Java Relational System (JRS):

- http://www.di.unipi.it/~albano/JRS/toStart.html
Selectivity of selection $\sigma_{\psi}(E)$

- The DBMS Catalog stores information about database tables and indexes.

$$s_f(A = v) = \frac{1}{N_{\text{key}}(A)}$$  \hspace{1cm} (1/10)

$$s_f(A > v) = \frac{\max(A) - v}{\max(A) - \min(A)}$$  \hspace{1cm} (1/3)

$$s_f(A < v) = \frac{v - \min(A)}{\max(A) - \min(A)}$$  \hspace{1cm} (1/3)

$$s_f(v_1 < A < v_2) = \frac{v_2 - v_1}{\max(A) - \min(A)}$$  \hspace{1cm} (1/4)
Selectivity of selection $\sigma_{\psi}(E)$

\[
s_f(A = B) = \frac{1}{\max(N_{\text{key}}(A), N_{\text{key}}(B))}
\]  

\[
s_f(\psi_1 \land \psi_2) = s_f(\psi_1) \times s_f(\psi_2)
\]

\[
s_f(\psi_1 \lor \psi_2) = s_f(\psi_1) + s_f(\psi_2) - s_f(\psi_1) \times s_f(\psi_2)
\]
Hystograms

105 records and A with 15 possible values in the range 17 to 31

Uniform distribution

NonUniform distribution
Selectivity of selection with histograms

• An Equi-Height histogram is used as an approximation of the actual distribution

• The active domain of A is divided into k intervals, containing a number of records of about $N_{rec}/k$

• For each interval $h_i$ are known:
  – $\min(h_i)$, $\max(h_i)$, $N_{key}(h_i)$, $N_{rec}(h_i)$
Equi-Height histograms
Equi-Height histograms

- $s_f(V = 24) = \frac{N_{rec}(V=24)}{N_{rec}}$
- $s_f \text{ real} = \frac{6}{105} = 0.057$
- $s_f \text{ uniform} = \frac{1}{15} = \frac{7}{105} = 0.066$
- $s_f \text{ histEqH} = \frac{(26/3)}{105} = 0.082$
Physical operators for tables and sort

- Operators for R:
  - TableScan (R)
  - SortScan (R, \{A_i\})
  - IndexScan (R, Idx)
  - IndexSequentialScan (R, Idx)

- Operator to sort (\tau \{A_i\}): 
  - Sort (O, \{A_i\})
Physical operators for $\pi_b\{A_i\}$

- **Project** ($O, \{A_i\}$): to project the records of $O$ without duplicates elimination
  - $C = C(O)$

- **IndexOnlyScan** ($R, Idx, \{A_i\}$):
  - $Idx$ an index on the attributes to project (or that contains them as prefix)
  - $C = N_{leaf}(Idx)$

- Is the result equivalent to a Project or to a Project+Distinct?
Physical ops for duplicate elimination

• Distinct (O): to eliminate duplicates from sorted records of O
  – C = C(O)

• Actually, we only need them to be grouped:
  – r_i=r_j and i<l<j => r_i=r_l=r_j

• HashDistinct(O): to eliminate duplicates from records of O;
  – C = ?
Hashdistinct

- **partitioning** phase:

- **duplicate elimination** phase ($h_r \neq h_p$):

\[ C = C(O) + 2 \ N_{pag}(O) \]
Sort

Hash
Result size: Distinct(O)

- If A is the only O attribute
  - $|Q| = N_{\text{key}}(A)$

- If $\{A_1, A_2, \ldots, A_n\}$ are the attributes
  - $|Q| = \min( |O|/2, \prod_i N_{\text{key}}(A_i) )$

- Why not just $\prod_i N_{\text{key}}(A_i)$?
Conclusion

• The sort method often preferred because produces a sorted result and because sort is heavily optimized

• DBMS:
  – Informix uses Hash
  – DB2, Oracle, Sybase ASE use Sort
  – SQL Server e Sybase ASIQ use both methods
Simple selection

• With no index, and data unsorted:
  – Relation scan with cost \( N_{\text{pag}}(R) \).

• With an index on selection attribute:
  – Use index to retrieve RIDs, then retrieve data records: \( C_l + C_D \).

• Cost depends on qualifying tuples (size of \( R \times s_f \)), and clustering

• General selection conditions…

```
SELECT * 
FROM Students 
WHERE City = 'PI'
```
Physical operators for selection

- **Filter** \((O, \psi)\)
  - \(C = C(O)\)
  - \(E_{\text{rec}} = s_f(\psi) \times N_{\text{rec}}(O)\)

- **IndexFilter** \((R, \text{Idx}, \psi)\)
  = \text{RidIndexFilter(Idx, } \psi) + \text{TableAccess(O, R)}
  - \(C = C_I + C_D\)
  - \(E_{\text{rec}} = s_f(\psi) \times N_{\text{rec}}(R)\)
Physical operators for selection

- **IndexSequentialFilter**($R$, $Idx$, $\psi$)
  - $C = s_f(\psi) \times N_{\text{leaf}}(Idx)$

- **IndexOnlyFilter**($R$, $Idx$, $\{A_i\}$, $\psi$)
  - $C = s_f(\psi) \times N_{\text{leaf}}(Idx)$

- **AndIndexFilter**($R$, $\{Idx_i, \psi_i\}$)
  - $C_l = \Sigma_i C_l(Idx_i)$
  - $C_D = \Phi(E_{\text{rec}}, N_{\text{pag}}(R))$
  - $E_{\text{rec}} = s_f(\psi) \times N_{\text{rec}}(R)$

- **OrIndexFilter**($R$, $\{Idx_i, \psi_i\}$)
Conjunctive selection in DBMSs

- Informix, DB2 use intersection of RID sets
- Oracle, Sybase ASIQ use bitmaps
- Oracle use also HashJoin with index (later) on the attribute RID
- Sybase ASE uses one index only
- SQL Server use index join (later)
Exercises

SELECT A, B FROM R WHERE (A BETWEEN 50 AND 100) AND B > 20;

Idx an index on A

SELECT A, B FROM R WHERE (A BETWEEN 50 AND 100) AND B > 20
ORDER BY A;

One index on A, B

SELECT DISTINCT A, B FROM R WHERE (A BETWEEN 50 AND 100) AND B > 20
ORDER BY B;

Two indexes on A and on B
Physical operators for grouping

• As for duplicate elimination:
  – Sorting
  – Hashing
  – Using an index on the grouping attributes

• (Duplicate elimination is grouping on all attributes with no aggregation)
Physical operators for ( \{A_i\} \gamma \{f_j\} )

- GroupBy (O, \{A_i\}, \{f_j\}): to group the records of O sorted on the \{A_i\}, computing functions in \{f_j\}.
  - \{f_j\} contains the aggregation functions present in the SELECT and HAVING clauses.
  - The operator returns records with attributes \{A_i\} \cup \{f_j\}
  - The records of O must be sorted on the \{A_i\}
  - Any permutation of \{A_i\} is ok
  - (Actually, they only need to be grouped on the \{A_i\})

- HashGroupBy (O, \{A_i\}, \{f_j\})

- C? Cardinality?
Computing aggregations

• Each aggregation function is an object o with methods start(), next(v), end()
• With the first record of a group: o.start()
• For each record of the group o.next(v) is invoked
• o.end() computes the final aggregate value.
Example

```
SELECT A, SUM(C) AS T
FROM R
WHERE A BETWEEN 50 AND 100
GROUP BY A
HAVING COUNT(*) > 1;
```

---

**Logical Tree**

```
\pi^b_{A,T}
\sigma_{N > 1}
\sigma_{...}
R
```

**Physical Tree**

```
TableScan (R)
Filter (...)
Sort ({A})
GroupBy ({A}, {SUM(C) AS T, COUNT(*) AS N})
Filter (N > 1)
Project ({A, T})
```

... = A BETWEEN 50 AND 100
Physical ops for join: nested loops

- foreach r in $O_E$ do
  - foreach s in $O_i$ do
    - if $r.r1 = s.s1$ then add $<r, s>$ to result

- $C = C(O_E) + E_{rec}(O_E) \times C(O_I)$
- $E_{rec} = s_f(C_j) \times E_{rec}(O_E) \times E_{rec}(O_I)$
- Inequality join conditions?
**Cost of nested loops**

\[
\text{C} = \text{Npag}(R) + \text{Nrec}(R) \times \text{Npag}(S) \approx \text{Npag}(R) \times \frac{\text{Nrec}(R)}{\text{Npag}(R)} \times \text{Npag}(S)
\]

With R **external**:

\[
\text{C} = \text{Npag}(R) + \text{Nrec}(R) \times \text{Npag}(S) \approx \text{Npag}(R) \times \frac{\text{Nrec}(R)}{\text{Npag}(R)} \times \text{Npag}(S)
\]

With S **external**:

\[
\text{C} = \text{Npag}(S) + \text{Nrec}(S) \times \text{Npag}(R) \approx \text{Npag}(S) \times \frac{\text{Nrec}(S)}{\text{Npag}(S)} \times \text{Npag}(R)
\]

Hence...

\[
\begin{align*}
\text{C} & \approx \text{Npag}(S) \times \frac{\text{Nrec}(S)}{\text{Npag}(S)} \times \text{Npag}(R) \\
& \approx \text{Npag}(S) \times \frac{\text{Nrec}(R)}{\text{Npag}(R)} \times \text{Npag}(S)
\end{align*}
\]
Page nested loops

C = \text{Npag}(R) + \text{Npag}(R) \times \text{Npag}(S)

The smallest relation is used as \textbf{external}
Block nested loops join

$$C_{BNL} = N_{pag}(R) + \left\lceil N_{pag}(R)/B \right\rceil \times N_{pag}(S)$$
Index nested loop

- Hyp: There is an index on the join column of the internal relation (S)

\[
\text{foreach } r \text{ in } R \text{ do}
\]

\[
\text{foreach } s \text{ in } \text{IndexFilter}(S, I, s.s1=r.r1) \text{ do}
\]

\[
\text{add } <r, s> \text{ to result}
\]
Index nested loop

foreach r in R do
    foreach s in \textbf{IndexFilter}(S,I,s1 =r.r1) do
        add <r, s> to result

\textbullet{} Cost for R join S:
\hspace{1em} C = N_{\text{pag}}(R) + N_{\text{rec}}(R) \times \text{CaWithIdx}(S)

\textbullet{} General Case:
\hspace{1em} C = C(O_E) + E_{\text{rec}}(O_E) \times (C_I+C_D)
Merge join

• Hyp: R and S are sorted on the join attribute, a key of the external relation
• $C = C(O_E) + C(O_I)$
• Inequality join conditions?

<table>
<thead>
<tr>
<th></th>
<th>ri</th>
<th>sj</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Hash join

• Partitioning (R and S)

• Matching
Hash join: cost

- Assume $N_{\text{pag}}(R)/B < B$ and uniformity
- $C = C(O_E) + C(O_I) + 2( N_{\text{pag}}(O_E) + N_{\text{pag}}(O_I) )$
- $C = (\log_B(N_{\text{pag}}(O_E)) \times 2 - 2) \times ( N_{\text{pag}}(O_E) + N_{\text{pag}}(O_I) )$
- $C(R \text{ join } S) = 3 \times ( N_{\text{pag}}(R) + N_{\text{pag}}(S) )$
- What if $N_{\text{pag}}(R) < B$?
Complex joins

• Join with a conjunctive condition
• Join with a disjunctive condition
Physical operators for join

- NestedLoop \((O_E, O_I, \psi_J)\)
- PageNestedLoop \((O_E, O_I, \psi_J)\)
- IndexNestedLoop \((O_E, O_I, \psi_J)\)
- MergeJoin \((O_E, O_I, \psi_J)\)
- HashJoin \((O_E, O_I, \psi_J)\)
Example

```
SELECT aR, Sum(cR)
FROM R, S
WHERE R.PkR=S.FkR AND cS=1000
GROUP BY aR
ORDER BY aR
```

\[ C_{\text{GroupBy}} = C_{\text{Sort}} = C_{\text{IndexNestedLoop}} + 2 \times N_{\text{pag}}(\text{Project}) \]

\[ N_{\text{pag}}(\text{Project}) = (E_{\text{Rec}}(\text{IndNestedLoop}) \times (L(aR) + L(cR))) / D_{\text{pag}} \]

\[ C_{\text{INL}} = C(\text{IndexFilterOnR}) + E_{\text{rec}}(\text{IndexFilterOnR}) \times C(\text{Filter}) \]

\[ E_{\text{rec}}(\text{INL}) = s_f(PkR = FkR) \times E_{\text{rec}}(\text{IndexFilterOnR}) \times E_{\text{rec}}(\text{Filter}) \]
Example

- \( C(\text{IndFilOnR}) = C_I + C_D \)
  - \( C_I = \left\lceil \frac{N_{\text{leaf}}(\text{ICR})}{N_{\text{key}}(\text{ICR})} \right\rceil \)
  - \( C_D = \Phi(\frac{N_{\text{rec}}(R)}{N_{\text{key}}(\text{ICR})}, N_{\text{pag}}(R)) \)
- \( E_{\text{rec}}(\text{IndFilOnR}) = \frac{N_{\text{rec}}(R)}{N_{\text{key}}(\text{ICR})} \)
- \( C(\text{IndFilOnS}) = C_I + C_D \)
  - \( C_I = \left\lceil \frac{N_{\text{leaf}}(\text{IFkR})}{N_{\text{key}}(\text{IFkR})} \right\rceil \)
  - \( C_D = \Phi(\frac{N_{\text{rec}}(S)}{N_{\text{key}}(\text{IFkR})}, N_{\text{pag}}(S)) \)
- \( E_{\text{rec}}(\text{Filter}) = s_f(cS=1000) \times N_{\text{rec}}(S) \)
Set operators

- Union($O_E, O_I$), Except($O_E, O_I$), Intersect($O_E, O_I$)
  - Operand sorted and without duplicates.
- HashUnion($O_E, O_I$), HashExcept($O_E, O_I$), HasIntersect($O_E, O_I$)
  - Using hash.
- UnionAll($O_E, O_I$)
  - trivial
- ExceptAll($O_E, O_I$), IntersectAll($O_E, O_I$)
  - Not always supported
Example

```
SELECT Name
FROM Employees
UNION
SELECT Name
FROM Professors;
```

```
SELECT Name
FROM Employees
UNION
SELECT Name
FROM Professors;
```
Summary

• Very few basic operators, hence the implementation of these operators can be carefully tuned.
• No universally superior technique for operators with many implementations
• We must consider available alternatives and select the best one: “Query optimization”
• ... the next topic