A (better?) Framework for Recycling Intermediates

Franco Solleza & Kewei Li
Why bother?

select col1 from table1 where col2 > 50

select col1 from table1 where col2 > 60

select col1 from table1 where col2 < 30

select col1 from table1 where 35 <= col2 < 45

select col1 from table1 where col2 > 35 and col3 < 40
Not that complicated
Not that complicated

select col1 from table1
where col2 > 50
Not that complicated

select col1 from table1
where col2 > 60

Scan only these portions of the data
Not that complicated

select col1 from table1
where col2 > 60
Not that complicated

select col1 from table1
where col2 < 30
Still Not that complicated

```sql
select col1 from table1
where col2 < 30
```

Not greater than 50
Still Not that complicated

select col1 from table1
where col2 < 30
Still Not that complicated

select col1 from table1
where col2 < 30

we can keep the < 50 instead of the > 50
Still Not that complicated

```sql
select col1 from table1
where col2 < 30
```

we can keep the < 50 instead of the > 50
Not that complicated?

select col1 from table1
where 35 <= col2 < 45

Hint: more than one way - which is better?
Pretty simple right?

```
select col1 from table1
where 35 \leq col2 < 45
```

Option 1:
NOT < 30

Cost:
Finding inverse of intermediate
Scans 7 pages
But there’s more than one way

```sql
select col1 from table1
where 35 <= col2 < 45
```

Option 2:

< 50

Cost:

Scans 7 pages
Not that complicated?

select col1 from table1
where 35 <= col2 < 45

Option 2:
NOT < 30
select col1 from table1
where 35 <= col2 < 45

Option 2:
NOT < 30
AND < 50

Cost:
Inverse of < 30
And with 50
Scan 3 pages
Adaptive
Adaptive
Adaptive

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Finding Intermediates

Options:

- Find the smallest superset of the query range
- Find the largest subset of the query range
- Find the smallest superset then:
  - exclude from the scan all that we know don’t qualify
  - skip from the scan all that we know do
Considerations

Cost model:
- What is a feasible cost model?

Admission and Eviction Policy
- How could we keep cache size manageable?

Finding and Composing Intermediates
- How can we minimize cost of finding intermediates?
Example of sub-space in problem space
Scan Technique Spaghetti - 100m tuples

Let’s always use bit vectors!
Let's always use bit vectors!

And make it single threaded below certain selectivity?
Multithreaded largely removes the branch misprediction costs?
Scan Technique Spaghetti - 100m tuples

Branching and no branching have no difference at low and high selectivity?
There may be some benefit!
All in vain?

Examples:

- What will fetch performance be like?
- A single NOT or AND operation is more expensive than a full scan
- Very implementation specific