Execution Plans

- an *execution plan* consists of query tree and algorithm for each relational algebra operation in the query tree

Query Optimization

Query optimization refers to choosing a reasonably good execution plan from the (generally) many possible ones.

Why generally many choices?

- multiple algorithm possibilities
  - may be multiple algorithms for carrying out a relational algebra operator
  - options for whether or not to use an index
- multiple query trees
  - multiple options for the order of application of relational algebra operations
  - may be more than one way to write a query
  - some constructs are declarative, with multiple ways to find results
    - e.g. EXISTS

Query Optimization

Query optimization refers to choosing a reasonably good execution plan from the (generally) many possible ones.

Why only a “reasonably good” execution plan and not the best?

- can be expensive to determine the best, or at least take longer than the time it saves

Query Optimization

Two main strategies:

- *heuristic* – apply rules-of-thumb for how to order operations
  - generally works well, but is not guaranteed to produce the best results
- *cost-based evaluation* – estimate the cost (e.g. number of disk blocks accessed) of different strategies, then choose best
  - need fast-to-compute cost functions
  - must limit the number of strategies considered

Heuristic Query Optimization

Process:

- query parser generates initial query tree corresponding to the query
- query tree is optimized according to heuristic rules, producing a new but equivalent query tree
  - *equivalent* = produces the same tuples
- query execution plan is generated by assigning algorithms for the relational algebra operations

Heuristic Query Optimization

What takes time?

- reading and writing blocks

We don't have control over the size of the files or the number of records we start or end with, but we do have control over the number and size of records in the intermediate results.

Heuristic: apply the operations that do the most to reduce the size of the intermediate results first.

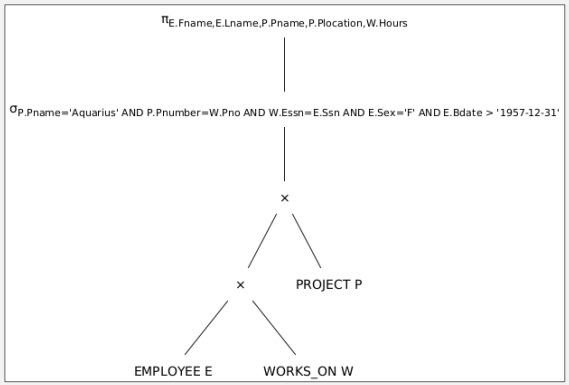
- SELECT reduces the number of rows (number of records)
- PROJECT reduces the number of columns (size of records)
- do the most restrictive SELECT, JOIN operations first
  - *most restrictive* = produces fewest rows

Heuristic Query Optimization Summary

1. Break up conjunctive SELECT conditions (ANDs) into a series of individual SELECTs.
2. Move each SELECT as far down as possible.
3. Combine CROSS PRODUCT and SELECT into JOIN.
4. Reorder consecutive SELECTs so the most selective are done first.
5. Reorder JOINs so the most restrictive operations are done first. (without introducing CROSS PRODUCTs)
6. Break up PROJECTs and move them as far down as possible.
7. Address opportunities for pipelining.

Initial Query Tree

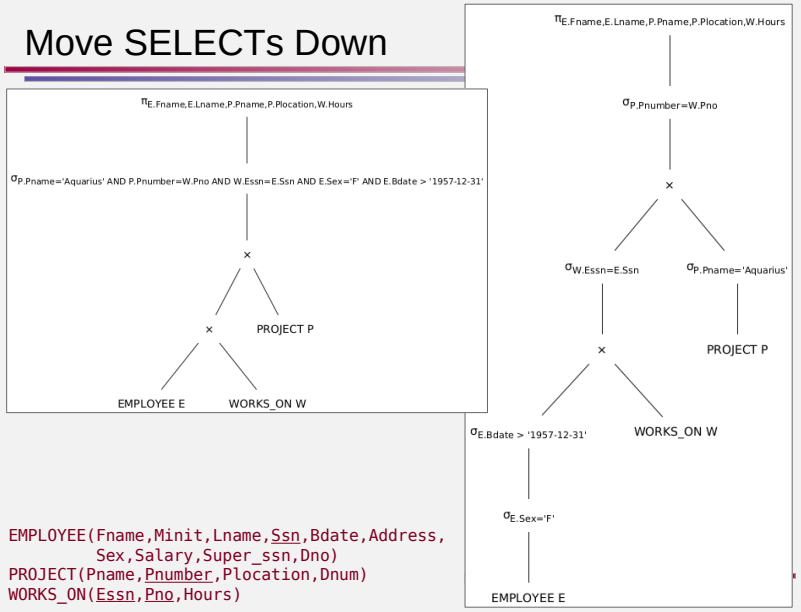
```
SELECT E.Fname,E.Lname,P.Pname,P.Plocation,W.Hours
FROM EMPLOYEE E, WORKS_ON W, PROJECT P
WHERE P.Pname='Aquarius' AND P.Pnumber=W.Pno AND
      W.Essn=E.Ssn AND E.Bdate > '1957-12-31' AND E.Sex='F'
```



Algorithm

1. Break up conjunctive SELECT conditions into a series of individual SELECT operations.
2. Move each SELECT as far down as possible.
   - SELECT is commutative (order of SELECTs doesn't matter)
   - SELECT can be moved below PROJECT
   - SELECT can be moved below JOIN and CROSS PRODUCT if the selection condition involves only attributes in one of the relations being joined
   - SELECT can be moved below UNION, INTERSECTION, and DIFFERENCE

Move SELECTs Down

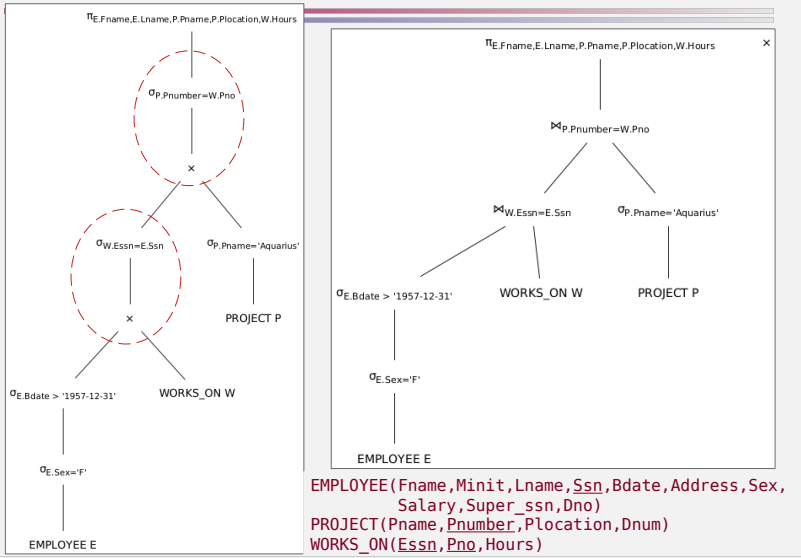



```
EMPLOYEE(Fname,Minit,Lname,Ssn,Bdate,Address,
         Sex,Salary,Super_ssn,Dno)
PROJECT(Pname,Pnumber,Plocation,Dnum)
WORKS_ON(Essn,Pno,Hours)
```

Algorithm

3. Combine CROSS PRODUCT and SELECT into JOIN.
   - requires SELECT condition to be a conjunction of equality comparisons

Transform CROSS PRODUCT Into JOIN




```
EMPLOYEE(Fname,Minit,Lname,Ssn,Bdate,Address,Sex,
         Salary,Super_ssn,Dno)
PROJECT(Pname,Pnumber,Plocation,Dnum)
WORKS_ON(Essn,Pno,Hours)
```

Algorithm

4. Reorder consecutive SELECTs so that the most selective are done first.
   - SELECT is commutative (order of SELECTs doesn't matter)
   - most selective can be based on the number of result tuples or the selectivity or rule of thumb (equality more selective than range)

Selectivity is the ratio of rows resulting from the SELECT to initial rows: $|\sigma_c(R)|/|R|$

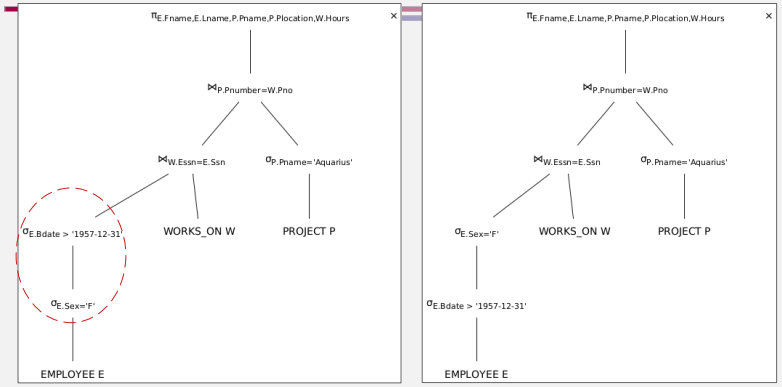
- can be estimated from DB catalog info
  - use number of distinct values to estimate for equality conditions
  - use range of values to estimate for range conditions

Database Catalog Info

table	column	# distinct values (d)	low value	high value
PROJECT	Pnumber	11	1	92
	Pname	11		
	Plocation	9		
	Dnum	5	1	8
WORKS_ON	Pno	11	1	92
	Essn	38		
EMPLOYEE	Ssn	40		
	Bdate	40	1927-11-10	1980-05-21
	Lname	37		
	Sex	2		

table	# records (r)
PROJECT	11
WORKS_ON	48
EMPLOYEE	40

Reorder SELECTs



$\sigma_{Bdate>'1957\text{-}12\text{-}31'}$(EMPLOYEE)
- date span is approx. 53 years, with 22 years covered by the range → assuming uniform distribution, selectivity is approx. 22/53 = 0.41
- number of result tuples = 0.41*40 = 16

$\sigma_{Sex='F'}$(EMPLOYEE)
- 2 distinct values → assuming uniform distribution, selectivity is 1/2 = 0.5
- number of result tuples = 0.5*40 = 20