COP 4710: Database Systems
Spring 2004

-Day 21 – March 24, 2004 –
Query Processing and Optimization

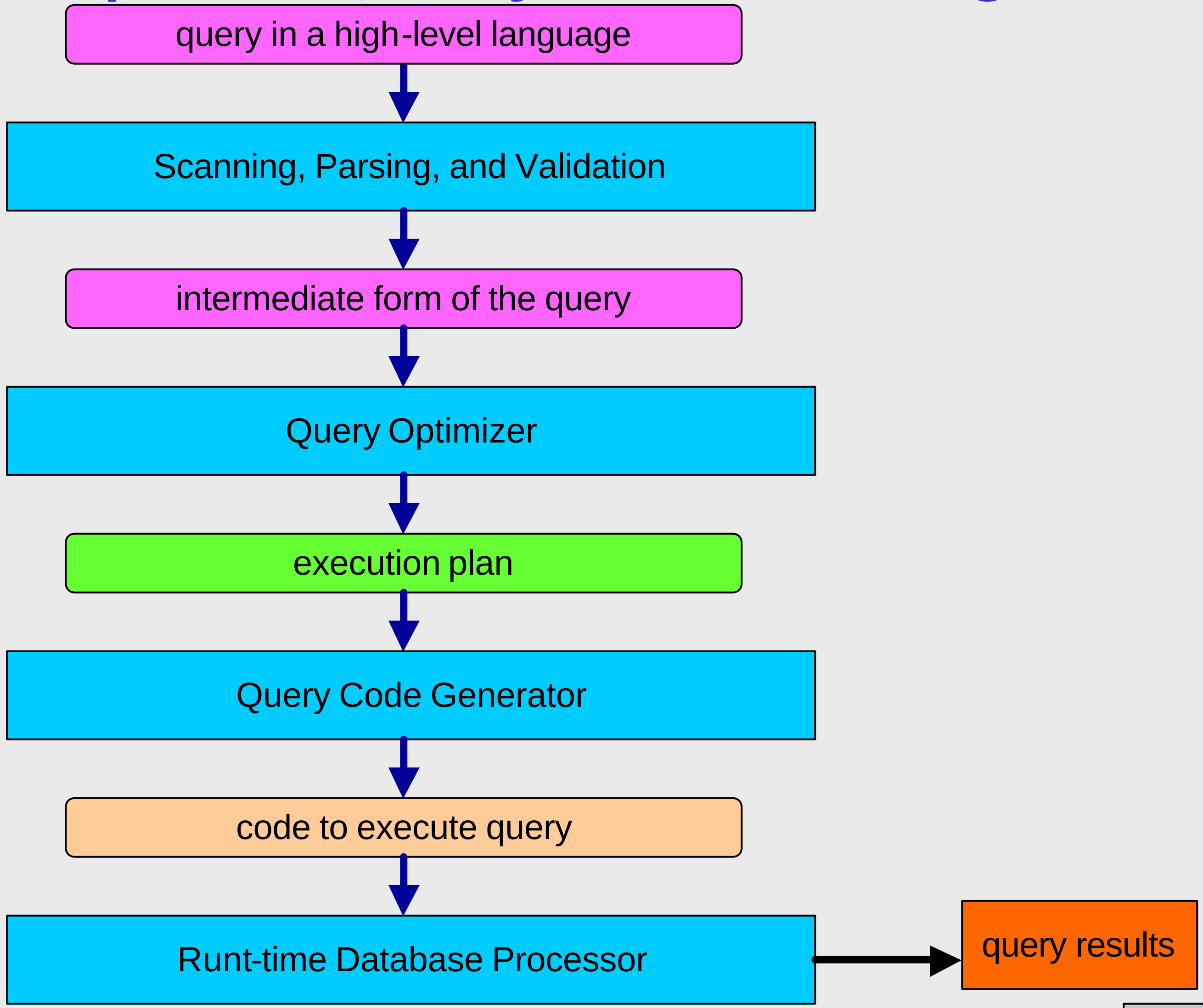
Instructor : Mark Llewellyn
markl@cs.ucf.edu
CC1 211, 823-2790
http://www.cs.ucf.edu/courses/cop4710/spr2004

School of Electrical Engineering and Computer Science
University of Central Florida

Query Processing and Optimization

- A query expresses in a high-level language like SQL must first be scanned, parsed, and validated.
- Once the above steps are completed, an internal representation of the query is created. Typically this is either a tree or graph structure, called a query tree or query graph.
- Using the query tree or query graph the RDBMS must devise an execution strategy for retrieving the results from the internal files.
- For all but the most simple queries, several different execution strategies are possible. The process of choosing a suitable execution strategy is called query optimization.

The Steps in Query Processing


query in a high-level language
Scanning, Parsing, and Validation
intermediate form of the query
Query Optimizer
execution plan
Query Code Generator
code to execute query
Runt-time Database Processor
query results

Query Optimization

- The term query optimization may be somewhat misleading. Typically, no attempt is made to achieve an optimal query execution strategy overall – merely a *reasonably efficient strategy*.
- Finding an optimal strategy is usually too time consuming except for very simple queries and for these it usually doesn't matter.
- Queries may be "hand-tuned" for optimal performance, but this is rare.
- Each RDBMS will typically maintain a number of general database access algorithms that implement basic relational operations such as select and join. Hybrid combinations of relational operations also typically exist.

Query Optimization (cont.)

- Only execution strategies that can be implemented by the DBMS access algorithms and which apply to the particular database in question can be considered by the query optimizer.

- There are two basic techniques that can be applied to query optimization:

  1. Heuristic rules: these are rules that will typically reorder the operations in the query tree for a particular execution strategy.

  2. Systematical estimation: the cost of various execution strategies are systematically estimated and the plan with the least "cost" is chosen. What constitutes cost can also vary. It could be a monetary cost, or it could be a cost in terms of time or other factors.

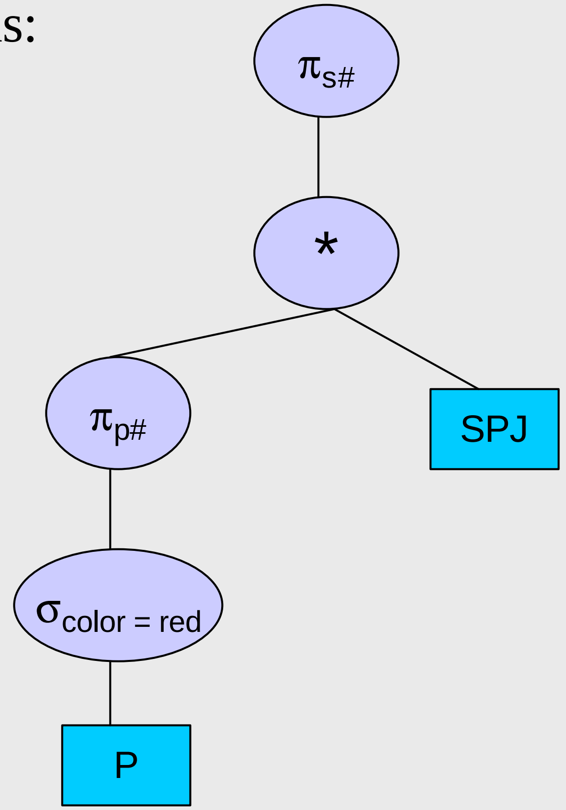
- Most query optimizers use a combination of both techniques.

Query Trees

- A query tree is a tree representation of a relational algebra expression which represents the operand relations as leaf nodes and the relational algebra operators as internal nodes.
- Execution of the query tree consists of executing and internal node operation whenever its operands are available and then replacing that internal node by the virtual relation which results from the execution of the operation.
- Execution terminates when the root node is executed and the resulting relation is produced.
- This technique is similar to what many compilers do for 3GLs like C.

Query Tree Example

- Consider the query: “list the supplier numbers for suppliers who supply a red part.” (this one should be really familiar by now!!)
- In relational algebra we have: $p_{s\#}(\text{spj} * (p_{p\#}(s_{\text{color='red'}}(\text{P}))))$
- The corresponding query tree is:



Query Trees

- There are usually several different ways to generate a relational algebra expression for a query. This should be quite obvious by now after doing the homework for the course.
- Since several different relational algebra expressions are possible for a given query, so too are there multiple query trees possible for the same query.
- The next page shows several different relational algebra expressions for a given query and the following couple of pages illustrate the possible query trees.

Query Expressions

- Query: list the names of those suppliers who ship both part numbers P1 and P2.

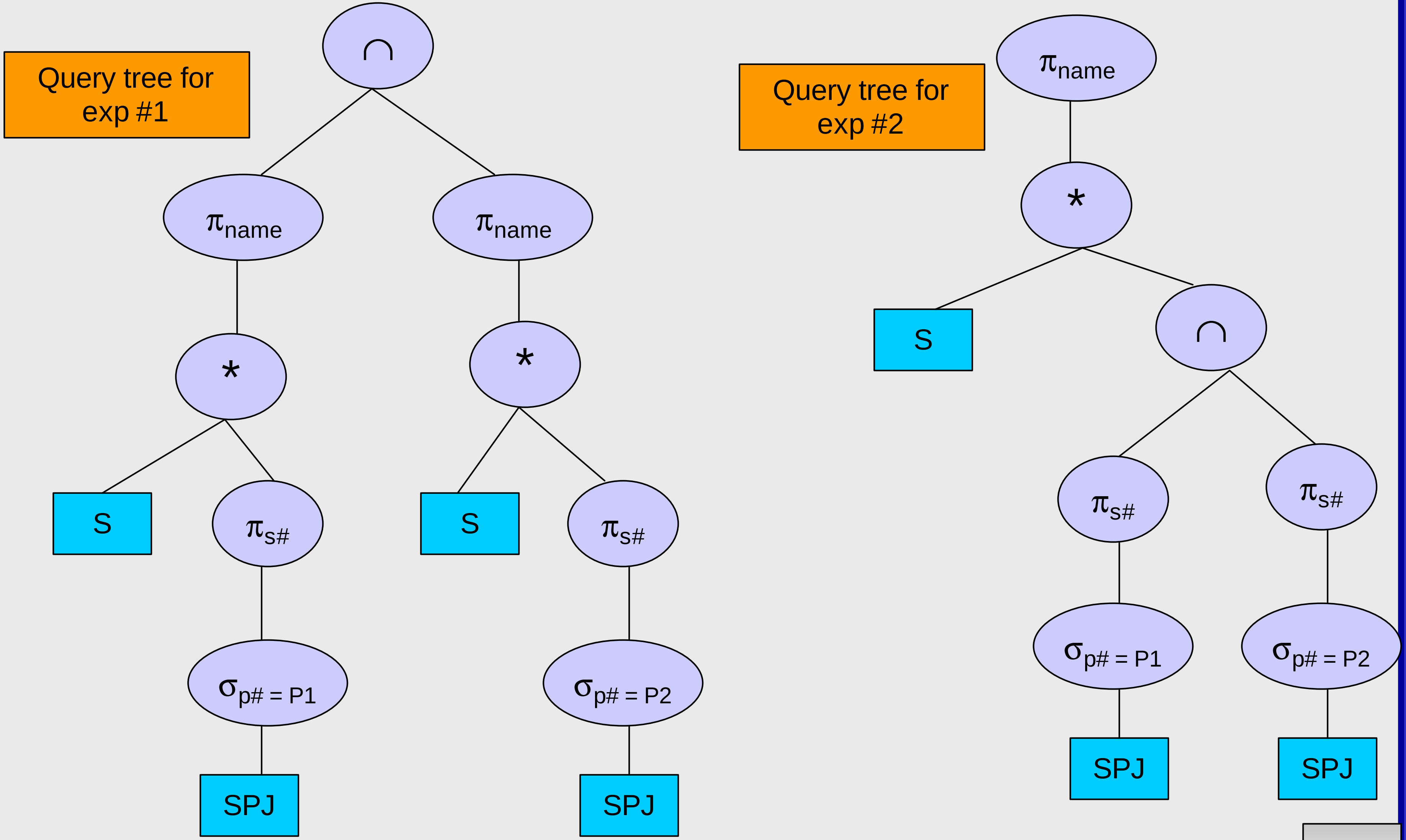
exp #1: $\left(p_{\text{name}}\left(\text{s} * \left(p_{\text{s\#}}\left(s_{\text{p\#=P1}}(\text{spj})\right)\right)\right)\right) \cap \left(p_{\text{name}}\left(\text{s} * \left(p_{\text{s\#}}\left(s_{\text{p\#=P2}}(\text{spj})\right)\right)\right)\right)$

exp #2: $p_{\text{name}}\left(\text{s} * \left(\left(p_{\text{s\#}}\left(s_{\text{p\#=P1}}(\text{spj})\right)\right) \cap \left(p_{\text{s\#}}\left(s_{\text{p\#=P2}}(\text{spj})\right)\right)\right)\right)$

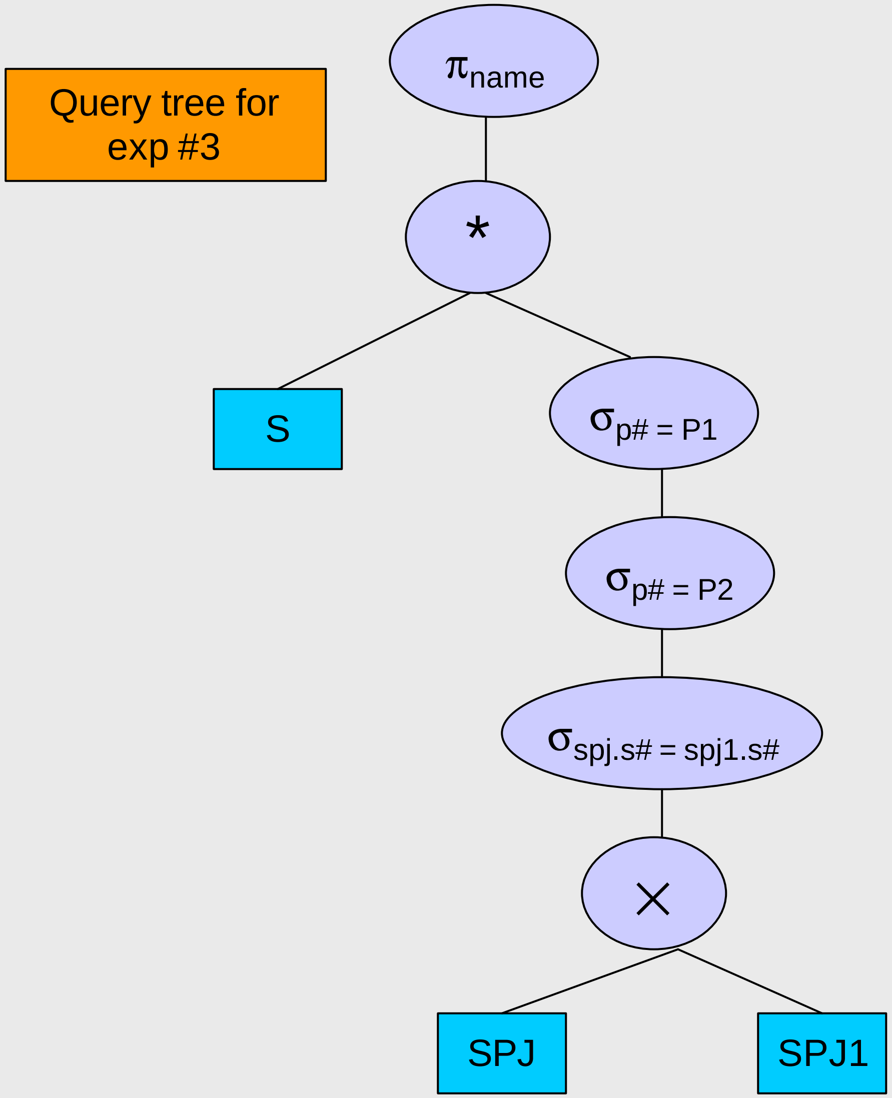
exp #3: $p_{\text{name}}\left(\text{s} * \left(p_{\text{s\#}}\left(s_{\text{spj.p\#=P1}}(\text{spj})\left(s_{\text{spj1.p\#=P2}}(\text{spj1})(\text{spj} \times \text{spj1})\right)\right)\right)\right)$

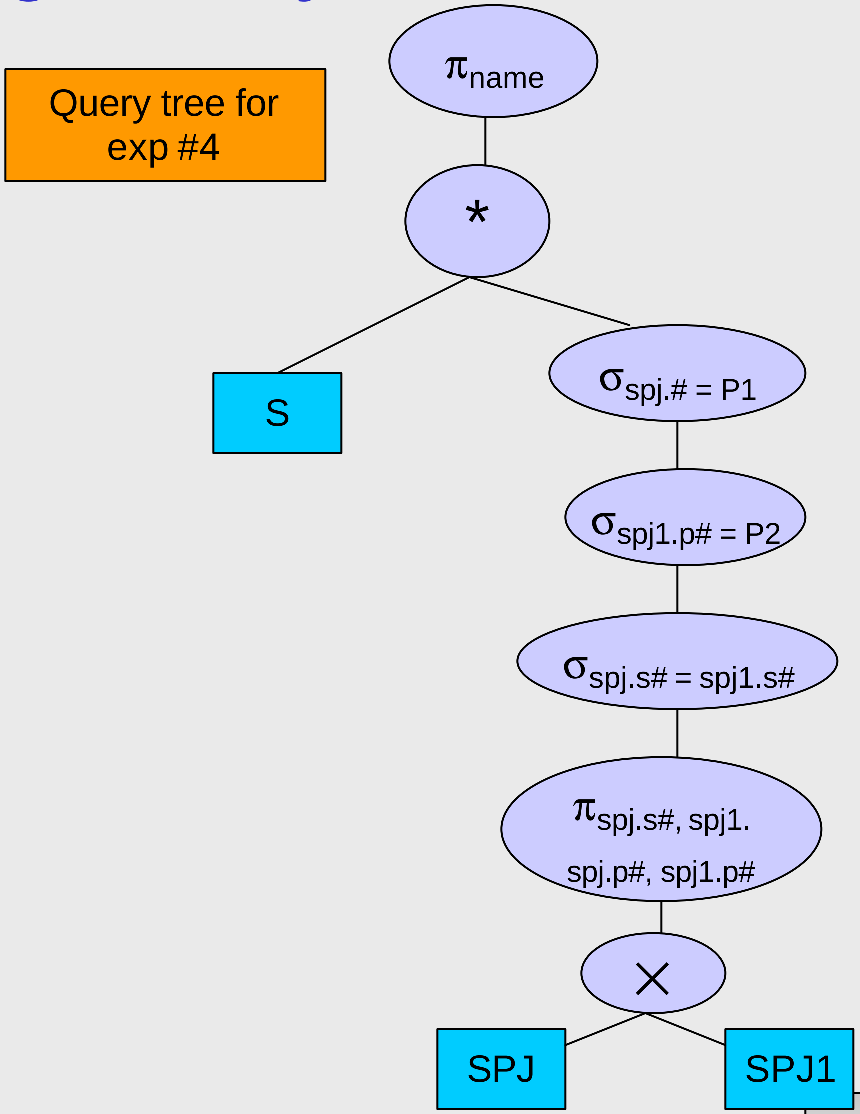
exp #4: $p_{\text{name}}\left(\text{s} * \left(s_{\text{spj.p\#=P1}}\left(s_{\text{spj1.p\#=P2}}\left(s_{\text{spj.s\#=spj1.s\#}}\left(p_{\text{spj.s\#,spl1.s\#,spj.p\#,spj1.p\#}}(\text{spj} \times \text{spj1})\right)\right)\right)\right)\right)$

Corresponding Query Trees

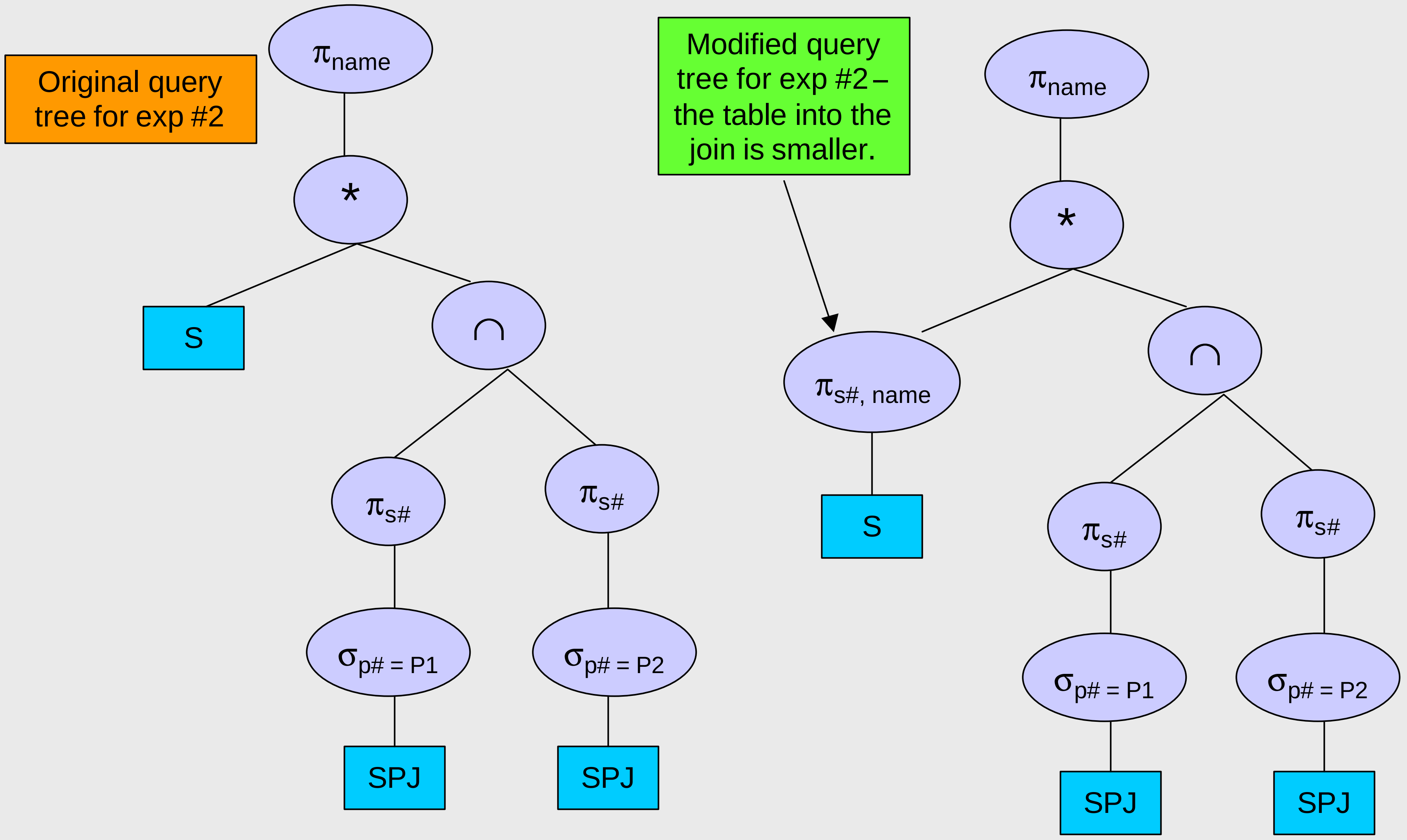


Corresponding Query Trees





Corresponding Query Trees



Basic Query Execution Algorithms

- For each operation (relational algebra operation, plus others) as well as combinations of operations, the DBMS will maintain one or more algorithms to execute the operation.
- Certain algorithms will apply to particular storage structures and access paths and thus can only be utilized if the underlying files involved in the operation include these access paths.
- Typically, the access paths will involve indices and/or hash tables, although other hybrid access paths are also possible.
- In the next few pages will examine some of these query execution strategies for the basic relational algebra operations.

Algorithms for Selection Operations

- There are many different options for Select operations based on the availability of access paths, indices, etc.
- Search algorithms for Select operations are one of two types:
  - index scans: search is directed from an index structure.
  - file scans: records are selected directly from the file structure.
- (FS1-linear search): Heap files typically are searched with a linear search algorithm.
- (FS2-binary search): Sequential files are typically searched with a binary or jump type of search algorithm.
- (IS3-primary index or hash key to extract single record): In these cases the selection condition involves an equality comparison on a key attribute for which a primary index has been created (or a hash key can be used.)

Algorithms for Selection Operations (cont.)

- (IS4-primary index or hash key to extract multiple records): In these cases the selection condition involves a non-equality based comparison (<, <=, >, >=) on a key attribute for which a primary index has been created. The primary index is used to find the record which satisfies the equality condition and then based upon this record, all other preceding (< or <=) or subsequent (> or >=) records are retrieved from the ordered file.
- (IS5-clustering index to extract multiple records): In these cases the selection condition involves an equality comparison on a non-key attribute which has a clustering index (a secondary index). The clustering index is used to retrieve all records which satisfy the selection condition.
- (IS6 – secondary index, B^+ tree): A selection condition with an equality comparison, a secondary index can be used to retrieve a single record if the indexing field is a key or to retrieve multiple records if the indexing field is not a key. Secondary indices can also be used for any of the comparison operators, not just equality.

Algorithms for Conjunctive Selections

- Conjunctive selections are selection conditions in which several conditions are logically AND’ed together.
- For simple (non-conjunctive) selection conditions, optimization basically means that you check for the existence of an access path on the attribute involved in the condition and use it if available, otherwise a linear search is performed.
- Query optimization for selection is most useful for conjunctive conditions whenever more than one of the participating attributes has an access path.
- The optimizer should choose the access path that retrieves the fewest records in the most efficient manner.

Algorithms for Conjunctive Selections (cont.)

- The overriding concern when choosing between multiple simple conditions in a conjunctive select condition is the selectivity of each condition.
- Selectivity is defined as:

$$\text{Selectivity} = \frac{\text{\# of records which satisfy the condition}}{\text{\# of records in the relation}}$$

- The smaller the selectivity the fewer the tuples the condition selects.
- Thus the optimizer should schedule the conjunctive selection comparisons so that the smallest selectivity conditions are applied first followed by the higher and higher selectivity values so that the last condition applied has the highest selectivity value.

Algorithms for Conjunctive Selections (cont.)

- Usually, exact selectivity values for all conditions are not available. However, the DBMS will maintain estimates for most if not all types of conditions and these estimates will be used by the optimizer.
- For example:
  - The selectivity of an equality condition on a key attribute of a relation r(R) is:

    $$\frac{1}{|r(R)|}$$

  - The selectivity of an equality condition on an attribute with n distinct values can be estimated by:

    $$\frac{\left(\frac{|r(R)|}{n}\right)}{|r(R)|} = \frac{1}{n}$$

    Assuming that the records are evenly distributed across the n distinct values, a total of |r(R)|/n records would satisfy an equality condition on this attribute.

Algorithms for Conjunctive Selections (cont.)

- (IS7-conjunctive selection): If an attribute is involved in any single simple condition in the conjunctive selection has an access path that permits the use of any of FS2 through IS6, use that condition to retrieve the records, then check if each retrieved record satisfies the remaining simple conditions in the conjunctive condition.
- (IS8-conjunctive selection using a composite index): If two or more attributes are involved in an equality condition and a composite index (or hash structure) exists for the combined fields – use the composite index directly.
- (IS9-conjunctive selection by intersection of record pointers): If secondary indices are available on any or all of the attributes involved in an equality comparison (assuming that the indices use record pointer and not block pointers), then each index is used to retrieve the record pointers that satisfy the individual simple conditions. The intersection of these record pointers is the set of tuples that satisfy the conjunction.