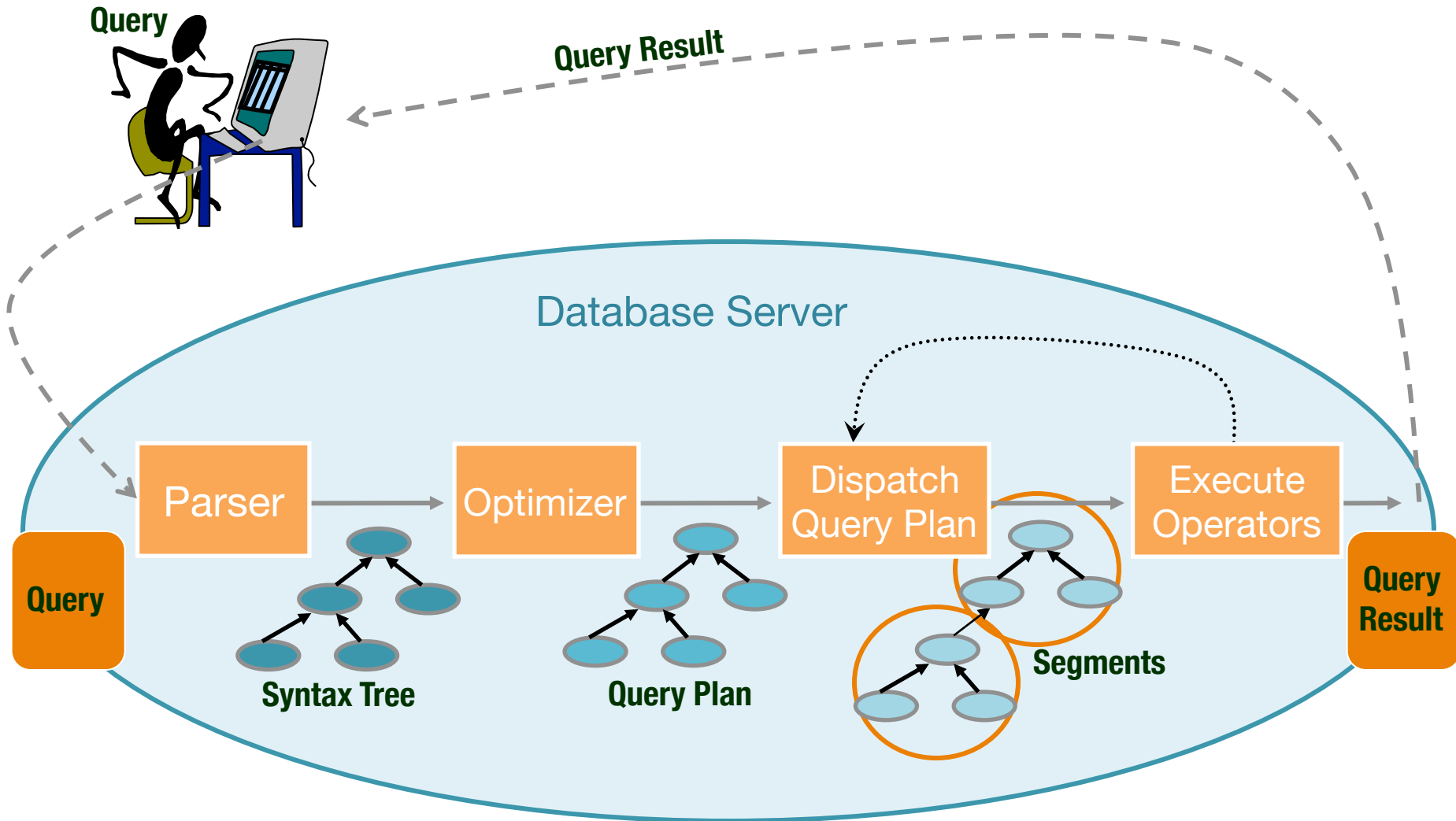


Handling Other Operations

Chapter 12 and 14

Query Execution Life-Cycle



Operator Evaluation

- How to implement common operators?
- ✓ • Selection
- ✓ • Join
- ➡ • Projection (optional DISTINCT)
 - Set Difference
 - Union
 - Aggregate operators (SUM, MIN, MAX, AVG)
 - GROUP BY

Projection

- `Select R.a, R.d`
 - Straightforward implementation!
- `Select DISTINCT R.a, R.d`
 - Remove attributes
 - Eliminate duplicates
- Algorithms for Projection `DISTINCT`:
 - Sorting: Sort on all the projected attributes
 - Pass 0: eliminate unwanted fields. Tuples in the sorted-runs may be smaller
 - Eliminate duplicates in the merge pass & sort
 - Hashing: Two phases
 - Partitioning
 - Duplicate elimination

Projection

- Sort-based approach
 - better handling of skew
 - result is sorted
 - Thus, more commonly used than hash-based approach
- Index-only scan
 - Projection attributes subset of index attributes
 - Apply projection techniques to data entries (much smaller!)
- If an ordered (i.e. tree) index contains all projection attributes as *prefix* of search key:
 - Retrieve index data entries in order (no sorting necessary)
 - Discard unwanted fields
 - Compare adjacent entries to eliminate duplicates (if required)

Operator Evaluation

- How to implement common operators?
- ✓ • Selection
- ✓ • Projection (optional DISTINCT)
- ✓ • Join
- ➡ • Set Difference
- Union
- Aggregate operators (SUM, MIN, MAX, AVG)
- GROUP BY

Set Operations

- \cap and \times special cases of join
- \cup and $-$ similar; we'll do \cup
 - Both require duplicate elimination
- Duplicate elimination algorithms for \cup :
 1. Sorting:
 - Sort both relations (on all attributes).
 - Merge sorted relations eliminating duplicates.
 2. Hashing:
 - Partition R and S
 - Build hash table for R_i .
 - Probe with tuples in S_i , add to table if not a duplicate

Operator Evaluation

- How to implement common operators?
 - ✓ • Selection
 - ✓ • Projection (optional DISTINCT)
 - ✓ • Join
 - ✓ • Set Difference
 - ✓ • Union
 - ➡ • Aggregate operators (SUM, MIN, MAX, AVG)
 - GROUP BY

Aggregates

- Sorting Approach
 - Sort on `GROUP BY` attributes (if any)
 - Scan sorted tuples, computing running aggregate
 - Min, Max
 - Count
 - Sum
 - Average: compute from sum and count
 - During scan, when the group by attribute changes (e.g. 2, 2, 2, **3**), output aggregate result

Aggregates

- Hashing Approach
 - Hash on GROUP BY attributes (if any)
 - Hash entry: grouped attributes + running aggregate
 - Scan tuples, probe hash table, update hash entry
 - Scan hash table, and output each hash entry
- Cost: Scan relation!

Aggregates

- Index
 - Without Grouping
 - Can use B+tree on aggregate attribute(s)
 - With grouping
 - B+tree on all attributes in `SELECT`, `WHERE` and `GROUP BY` clauses
 - Index-only scan
 - If group-by attributes prefix of search key
=> data entries/tuples retrieved in group-by order
 - Else => get data entries and then use a sort or hash aggregate algorithm

Summary

- Various algorithms to choose from for each operator:
 - Selection
 - Join
 - Simple / Page / Block Nested Loops
 - Merge-Join
 - Hash Join (to be continued)
 - Projection (optional DISTINCT)
 - Set Difference
 - Union
 - Aggregate operators (SUM, MIN, MAX, AVG)
 - GROUP BY

Optional Exercises

- 12.1 (1-4), 12.3, 12.5
- 13.1, 13.3
- 14.1 (2, 3, 4, 6, 7, 8, 9, 10), 14