CIS 560 – Database System Concepts

Lecture 25

Query Processing

November 1, 2013

Credits for slides: Chang, Ullman, Whitehead.

Outline

Last:

- Indexes and B-trees 14.1-14.2

Today:

- Query processing

Next:

- Query execution 15.1-15.6
- Query optimization 16

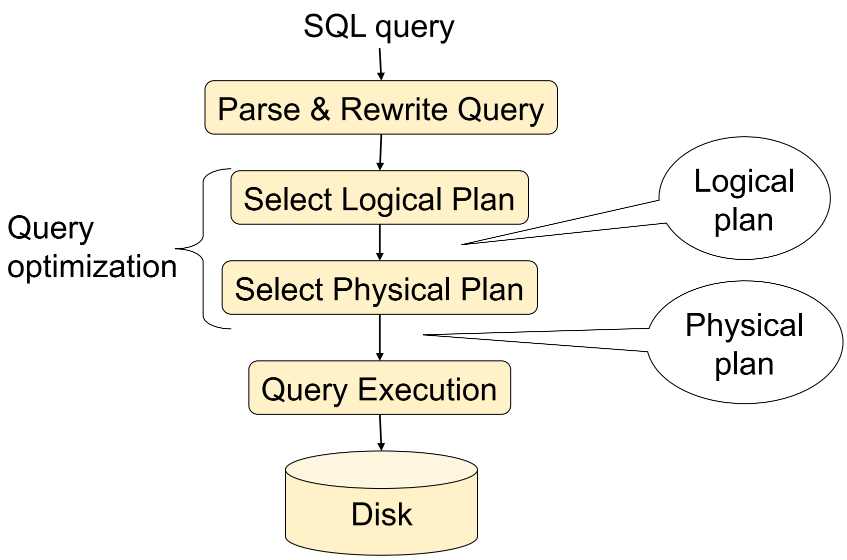
2

Planning

- Assignment 7 (concurrency control) due 11/1
- Assignment 8 (indexes) due 11/8
- Assignment 9 (query optimization) due 11/15
- Exam 2 (assignments 6-9) – 11/20
- Project assignment due 11/22
- Quiz from special topics – 12/06
- Project presentations 12/9, 12/11, 12/13
- Project reports – finals weeks

3

Steps of the Query Processor



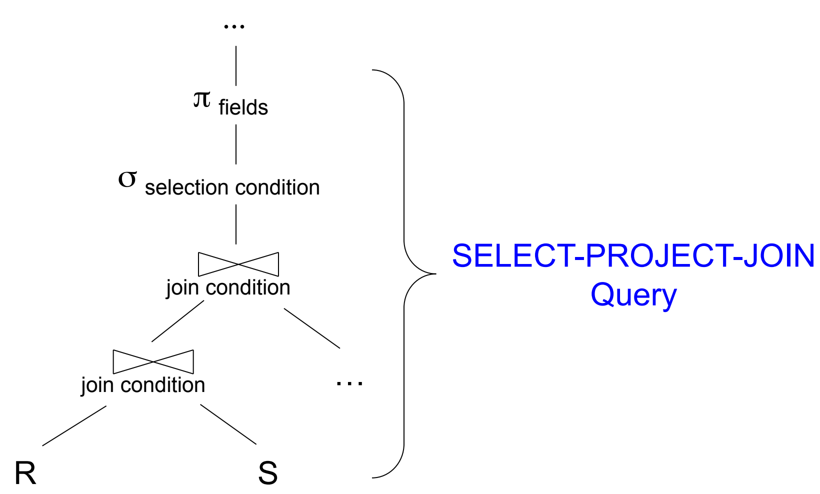
Query Optimization

- **Step 3: Query optimization**
  - Finds an efficient query plan for executing the query
- A **query plan** is
  - **Logical query plan**: an extended relational algebra tree
  - **Physical query plan**: with additional annotations at each node
    - Access method to use for each relation
    - Implementation to use for each relational operator

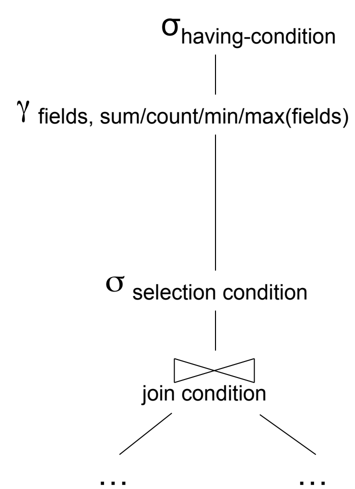
Query Block

- Most optimizers operate on individual query blocks
- A query block is an SQL query with **no nesting**
  - **Exactly one**
    - SELECT clause
    - FROM clause
  - **At most one**
    - WHERE clause
    - GROUP BY clause
    - HAVING clause

Typical Plan for Block (1/2)



Typical Plan For Block (2/2)

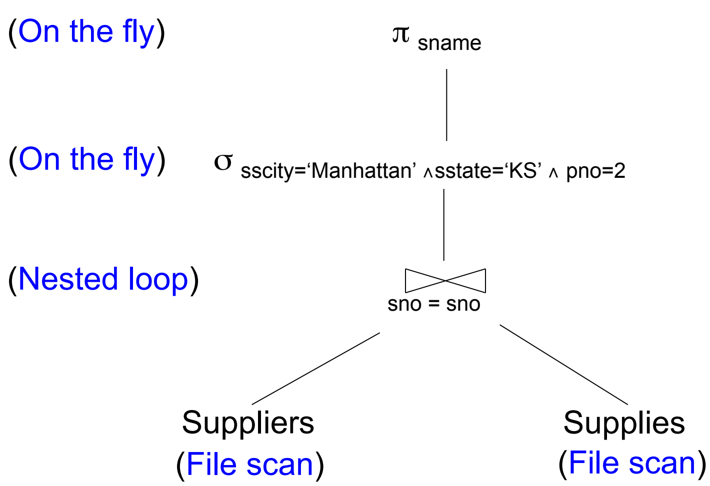


Physical Query Plan

- Logical query plan with extra annotations
- **Access path selection** for each relation
  - Use a file scan or use an index
- **Implementation choice** for each operator
- **Scheduling decisions** for operators

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

Physical Query Plan



Final Step in Query Processing

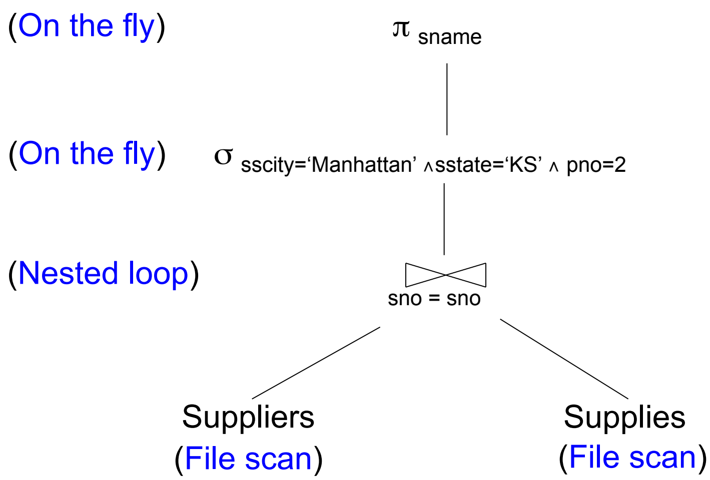
- **Step 4: Query execution**
  - How to synchronize operators?
  - How to pass data between operators?
- What techniques are possible?
  - One thread per query
  - Iterator interface
  - Pipelined execution
  - Intermediate result materialization

Iterator Interface

- Each **operator implements this interface**
- Interface has only three methods
- **open()**
  - Initializes operator state
  - Sets parameters such as selection condition
- **get_next()**
  - Operator invokes get_next() recursively on its inputs
  - Performs processing and produces an output tuple
- **close()**: cleans-up state

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

Pipelined Execution

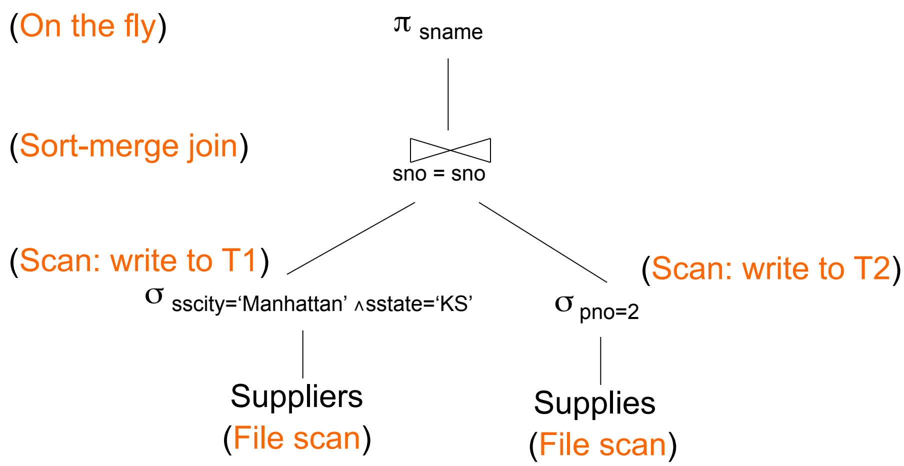


Pipelined Execution

- Applies parent operator to tuples directly as they are produced by child operators
- Benefits
  - No operator synchronization issues
  - Saves cost of writing intermediate data to disk
  - Saves cost of reading intermediate data from disk
  - Good resource utilizations on single processor
- This approach is used whenever possible

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

Intermediate Tuple Materialization



Intermediate Tuple Materialization

- Writes the results of an operator to an intermediate table on disk

No direct benefit but:

- Necessary when data is larger than main memory
- Necessary when operator needs to examine the same tuples multiple times

Physical Operators

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators (operator algorithms), with a focus on join

Why Learn About Operator Algorithms?

- Implemented in commercial DBMSs
  - DBMSs implement different subsets of known algorithms
- Good algorithms can greatly improve performance
- Need to know about physical operators to understand query optimization

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

Join Physical Operators

Logical operator:

Supplies(sno,pno,price) $\bowtie_{pno=pno}$ **Part(pno,pname,psize,pcolor)**

Three physical operators for the join, assuming the tables are in main memory.

Nested Loop Join
Merge join
Hash join

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

1. Nested Loop Join

```
for S in Supplies do {
  for  P in Part do {
      if (S.pno == P.pno) output(S,P);
  }
}
```

Supplies = *outer relation*
Part = *inner relation*
Note: sometimes terminology is switched

20

It's more complicated…

- Each **operator implements this interface**
- **open()**
- **get_next()**
- **close()**

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

Nested Loop Join Revisited

```
open ( ) {
  Supplies.open( );
  Part.open( );
  S = Supplies.get_next( );
}
```

```
close ( ) {
  Supplies.close ( );
  Part.close ( );
}
```

```
get_next( ) {
  repeat {
    P= Part.get_next( );
    if (P== NULL)
      { Part.close();
        S= Supplies.get_next( );
        if (S== NULL) return NULL;
        Part.open( );
        P= Part.get_next( );
      }
  } until (S.pno == P.pno);
  return (S, P)
}
```

ALL operators need to be implemented this way !

Hash-based Join

- R $\bowtie$ S
- Main memory *hash-based join*:
  - Scan S, build buckets in main memory
  - Then scan R and join

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

2. Hash Join (main memory)

Build phase

```
for S in Supplies do  insert(S.pno, S);

for P in Part do {
   LS = find(P.pno);
   for S in LS do { output(S, P); }
}
```

Probing

Supplies=outer
Part=inner

Recall: need to rewrite as open, get_next, close

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)

3. Merge Join (main memory)

```
Part1 = sort(Part, pno);
Supplies1 = sort(Supplies, pno);
P=Part1.get_next(); S=Supplies1.get_next();

While (P!=NULL and S!=NULL) {
   case:
      P.pno > S.pno:    P = Supplies1.get_next( );
      P.pno < S.pno:    S = Part1.get_next();
      P.pno == S.pno { output(P,S);
                       S = Supplies1.get_next();
                     }
}
```

Summary

- Three algorithms for main memory join:
  - Nested loop join
  - Hash join
  - Merge join

If $|R| = m$ and $|S| = n$, what is the asymptotic complexity for computing $R \bowtie S$?

Other Main Memory Algorithms

- Grouping: $\gamma(R)$
  - Nested loop
  - Hash table
  - Sorting
- Duplicate elimination: $\delta(R)$
  - *Exactly* the same algorithms (why?)

How do these algorithms work, and what are their complexities?