

Yannis Papakonstantinou, CSE232, UCSD

Query Processing

- The query processor turns user queries and data modification commands into a sequence of operations (or algorithm) on the database
 - from high level queries to low level commands
- Decisions taken by the query processor
 - Which of the algebraically equivalent forms of a query will lead to the most efficient algorithm?
 - For each algebraic operator what algorithm should we use to run the operator?
 - How should the operations pass data from one to the other? (eg. main memory buffers, disk buffers)

1

Yannis Papakonstantinou, CSE232, UCSD

Rough Architecture of the Query Processor

```

    graph TD
      A[SQL query] --> B[Parser]
      B --> C["Initial logical query plan (algebraic expression)"]
      C --> D[Logical Plan Generator/Chooser]
      D --> E["Candidate logical query plans"]
      E --> F[Physical Plan Generator/Chooser]
      F --> G["Optimal physical query plan (algebraic expression) annotated with details on (1) how each operator is implemented and (2) how data pass from one operator to the next"]
  
```

2

Yannis Papakonstantinou, CSE232, UCSD

Query Processing and Optimization: The Journey of a Query Revisited

SELECT Theater
FROM Movie, Schedule
WHERE
Movie.Title=Schedule.Title
AND Actor="Winger"

Parser → $\pi_{Theater}$ $\sigma_{Movie.Title=Schedule.Title \wedge Actor="Winger"}$ *Initial logical query plan*

Logical Plan Generator applies Algebraic Rewriting → $\pi_{Theater}$ $\sigma_{Actor="Winger"}$ *Another logical query plan*

Logical Plan Generator → $\pi_{Theater}$ $\sigma_{Actor="Winger"}$ $\sigma_{M.Title=S.Title}$ *Another logical query plan*

JOIN → $\pi_{Theater}$ $\sigma_{Actor="Winger"}$ $\sigma_{M.Title=S.Title}$ *Another logical query plan*

Next Page

3

Yannis Papakonstantinou, CSE232, UCSD

The Journey of a Query cont'd: Summary of Logical Plan Generator

4th logical query plan

- 4 logical query plans created
- algebraic rewritings were used for producing the candidate logical query plans
- the last one is the winner (at least, cannot be a big loser)
- in general, multiple logical plans may "win" eventually

if cond refers only on S

4

Yannis Papakonstantinou, CSE232, UCSD

The Journey of a Query Continues at the Physical Plan Generator

Physical Plan Generator chooses execution primitives and data passing

index on Actor, tables Schedule sorted on Title, $\pi_{Theater}$ SORT-MERGE $\sigma_{S.Title=M.Title}$

index on Actor and Title, unsorted tables, tables >> memory

Physical Plan 1: $\pi_{Theater}$ LEFT INDEX $\sigma_{S.Title=M.Title}$ $\sigma_{Actor="Winger"}$

Physical Plan 2: $\pi_{Theater}$ SORT-MERGE $\sigma_{S.Title=M.Title}$ INDEX $\sigma_{Actor="Winger"}$

More than one plans may be generated by choosing different primitives

5

Yannis Papakonstantinou, CSE232, UCSD

More Than One Plans May be Generated and Evaluated

Transform

Transform

6

Yannis Papakonstantinou, CSE232, UCSD

Issues in Query Processing and Optimization

- Generate Plans
 - systematically transform expressions ①
 - employ execution primitives for computing relational algebra operations ②
- Estimate Cost of Generated Plans ②
 - Statistics
- “Smart” Search of the Space of Possible Plans ③
 - always do the “good” transformations (relational algebra optimization)
 - prune the space (e.g., System R)
- Often the above steps are mixed

7

Yannis Papakonstantinou, CSE232, UCSD

Algebraic Operators: A Bag version

- *Union of R and S*: a tuple t is in the result as many times as the sum of the number of times it is in R plus the times it is in S
- *Intersection of R and S*: a tuple t is in the result the minimum of the number of times it is in R and S
- *Difference of R and S*: a tuple t is in the result the number of times it is in R minus the number of times it is in S
- $\mathbf{d}R$ converts the bag R into a set
 - SQL’s $R \text{ UNION } S$ is really $\mathbf{d}R \dot{\cup} S$
- **Example:** Let $R=\{A,B,B\}$ and $S=\{C,A,B,C\}$. Describe the union, intersection and difference... 8

Yannis Papakonstantinou, CSE232, UCSD

Extended Projection

- We extend the relational project π_A as follows:
 - The attribute list may include $x@y$ in the list A to indicate that the attribute x is renamed to y
 - Arithmetic or string operators on attributes are allowed. For example,
 - $a+b@x$ means that the sum of a and b is renamed into x .
 - $c||d@y$ concatenates the result of c and d into a new attribute named y
- The result is computed by considering each tuple in turn and constructing a new tuple by picking the attributes names in A and applying renamings and arithmetic and string operators
- **Example:** 9

Yannis Papakonstantinou, CSE232, UCSD

An Alternative Approach to Arithmetic and Other 1-1 Computations

- Special purpose operators that for every input tuple they produce one output tuple
 - $MULT_{A,B \rightarrow C}R$: for each tuple of R , multiply attribute A with attribute B and put the result in a new attribute named C .
 - $PLUS_{A,B \rightarrow C}R$
 - $CONCAT_{A,B \rightarrow C}R$
- **Exercise:** Write the above operators using extended projection. Assume the schema of R is $R(A,B,D,E)$. 10

Yannis Papakonstantinou, CSE232, UCSD

Product and Joins

- *Product of R and S ($R \times S$):*
 - If an attribute named a is found in both schemas then rename one column into $R.a$ and the other into $S.a$
 - If a tuple r is found n times in R and a tuple s is found m times in S then the product contains nm instances of the tuple rs
- Joins
 - *Natural Join* $R \bowtie S = \pi_A \sigma_C(R \times S)$ where
 - C is a condition that equates all common attributes
 - A is the concatenated list of attributes of R and S with no duplicates
 - you may view the above as a rewriting rule
 - *Theta Join*
 - arbitrary condition involving attributes

11

Yannis Papakonstantinou, CSE232, UCSD

Grouping and Aggregation

- Operators that combine the *GROUP-BY* clause with the aggregation operator ($AVG, SUM, MIN, MAX, \dots$)
- $SUM_{GroupbyList, GroupedAttribute@ResultAttribute} R$ corresponds to


```
SELECT GroupbyList,
       SUM(GroupedAttribute) AS ResultAttribute
FROM R
GROUP BY GroupbyList
```
- Similar for $AVG, MIN, MAX, COUNT, \dots$
- Note that $\mathbf{d}R$ could be seen as a special case of grouping and aggregation
- **Example** 12

Yannis Papakonstantinou, CSE232, UCSD

Algebraic Rewritings: Commutativity and Associativity

Commutativity

Associativity

Question 1: Do the above hold for both sets and bags?
Question 2: Do commutativity and associativity hold for arbitrary Theta Joins?

13

Yannis Papakonstantinou, CSE232, UCSD

Algebraic Rewritings: Commutativity and Associativity (2)

Commutativity

Associativity

Question 1: Do the above hold for both sets and bags?
Question 2: Is difference commutative and associative?

14

Yannis Papakonstantinou, CSE232, UCSD

Algebraic Rewritings for Selection: Decomposition of Logical Connectives

15

Yannis Papakonstantinou, CSE232, UCSD

Algebraic Rewritings for Selection: Decomposition of Negation

Question	Complete
$\sigma_{cond1 \text{ AND NOT } cond2} R$	$\sigma_{cond1} R \text{ AND NOT } \sigma_{cond2} R$
$\sigma_{\text{NOT } cond2} R$	$R \text{ AND NOT } \sigma_{cond2} R$
$\sigma_{cond1 \text{ OR NOT } cond2} R$	$\sigma_{cond1} R \cup \sigma_{\text{NOT } cond2} R$

16

Yannis Papakonstantinou, CSE232, UCSD

Pushing the Selection Thru Binary Operators: Union and Difference

Union

Difference

Exercise: Do the rule for intersection

17

Yannis Papakonstantinou, CSE232, UCSD

Pushing Selection thru Cartesian Product and Join

Exercise: Do the rule for theta join

18

Yannis Papakonstantinou, CSE232, UCSD

Pushing Simple Projections Thru Binary Operators

A projection is simple if it only consists of an attribute list

Union

Question 1: Does the above hold for both bags and sets?
Question 2: Can projection be pushed below intersection and difference?
 Answer for both bags and sets.

19

Yannis Papakonstantinou, CSE232, UCSD

Pushing Simple Projections Thru Binary Operators: Join and Cartesian Product

Where B is the list of R attributes that appear in A .
Similar for C .

Question: What is B and C ?

Exercise: Write the rewriting rule that pushes projection below theta join.

20

Yannis Papakonstantinou, CSE232, UCSD

Projection Decomposition

21

Yannis Papakonstantinou, CSE232, UCSD

Some Rewriting Rules Related to Aggregation: SUM

- $s_{cond} \hat{U} SUM_{GroupbyList; GroupedAttribute \otimes ResultAttribute}^R$
 if $cond$ involves only the $GroupbyList$
- $SUM_{GL; GA \otimes RA}(\hat{R} \hat{E} S)$
 $\hat{U} PLUS_{RA1, RA2: RA}((SUM_{GL; GA \otimes RA1} R) \triangleright \triangleleft (SUM_{GL; GA \otimes RA2} S))$
- $SUM_{GL2; RA1 \otimes RA2} SUM_{GL1; GA \otimes RA1} R \hat{U} SUM_{GL2; GA \otimes RA2} R$
 – **Question:** does the above hold for both bags and sets?

22

Yannis Papakonstantinou, CSE232, UCSD

Remarks

- In general there are many operators and many rewriting rules once we go beyond standard relational algebra.
 - Extensibility of the optimizer becomes paramount
- Some of the rules cannot be introduced in the algebraic rewriter as they are because they may lead to non-terminating derivations or unacceptable exponential explosions

23

Yannis Papakonstantinou, CSE232, UCSD

Exercises

- Write the following axioms for natural join
 - push join under union(s)
 - push a selection/projection under join
 - commutativity and associativity of join
- Prove the axioms you wrote using axioms given in the notes

24

Yannis Papakonstantinou, CSE232, UCSD

Algorithms for Relational Algebra Operators

- Three primary techniques
 - Sorting
 - Hashing
 - Indexing
- Three degrees of difficulty
 - data small enough to fit in memory
 - too large to fit in main memory but small enough to be handled by a “two-pass” algorithm
 - so large that “two-pass” methods have to be generalized to “multi-pass” methods (quite unlikely nowadays)

25

Yannis Papakonstantinou, CSE232, UCSD

Computation Model

- There are M main memory buffers.
 - Each buffer has the size of a disk block
- The input relation is read one block at a time.
- The cost is the number of blocks read.
- If B consecutive blocks are read the cost is B/d .
- The output buffers are not part of the M buffers mentioned above.
 - *Pipelining* allows the output buffers of an operator to be the input of the next one.
 - We do not count the cost of writing the output.

26

Yannis Papakonstantinou, CSE232, UCSD

Notation

- $B(R)$ = number of blocks that R occupies
- $T(R)$ = number of tuples of R
- $V(R, [a_1, a_2, \dots, a_n])$ = number of distinct tuples in the projection of R on a_1, a_2, \dots, a_n

27

Yannis Papakonstantinou, CSE232, UCSD

One-Pass Main Memory Algorithms for Unary Operators

- Assumption: Enough memory to keep the relation
- Projection and selection:
 - Scan the input relation R and apply operator one tuple at a time
 - Cost depends on
 - clustering of R
 - whether the blocks are consecutive
- Duplicate elimination and aggregation
 - create one entry for each group and compute the aggregated value of the group
 - it becomes hard to assume that CPU cost is negligible
 - main memory data structures are needed

28

Yannis Papakonstantinou, CSE232, UCSD

One-Pass Nested Loop Join

- Assume $B(R)$ is less than M
- Tuples of R should be stored in an efficient lookup structure
- **Exercise:** Find the cost of the algorithm below

```

for each block Br of R do
  store tuples of Br in main memory
for each each block Bs of S do
  for each tuple s of Bs
    join tuples of s with matching tuples of R
  
```

29

Yannis Papakonstantinou, CSE232, UCSD

Generalization of Nested-Loops

```

for each chunk of M-1 blocks Br of R do
  store tuples of Br in main memory
for each each block Bs of S do
  for each tuple s of Bs
    join tuples of s with matching tuples of R
  
```

Exercise: Compute cost

30

Yannis Papakonstantinou, CSE232, UCSD

Simple Sort-Merge Join

- Assume natural join on C
- Sort R on C using the two-phase multiway merge sort
 - if not already sorted
- Sort S on C
- Merge (opposite side)
 - assume two pointers Pr, Ps to tuples on disk, initially pointing at the start
 - sets R', S' in memory
- Remarks:
 - Very low average memory requirement during merging (but no guarantee on how much is needed)
 - **Cost:**

```

while Pr!=EOF and Ps!=EOF
  if *Pr[C] == *Ps[C]
    do_cart_prod(Pr,Ps)
  else if *Pr[C] > *Ps[C]
    Ps++
  else if *Ps[C] > *Pr[C]
    Pr++

function do_cart_prod(Pr,Ps)
  val=*Pr[C]
  while *Pr[C]==val
    store tuple *Pr in set R'
    while *Ps[C]==val
      store tuple *Ps in set S'
    output cartesian product
      of R' and S'
  
```

31

Yannis Papakonstantinou, CSE232, UCSD

Efficient Sort-Merge Join

- Idea: Save two disk I/O's per block by combining the second pass of sorting with the "merge".
- Step 1: Create sorted sublists of size M for R and S
- Step 2: Bring the first block of each sublist to a buffer
 - assume no more than M sublists in all
- Step 3: Repeatedly find the least C value c among the first tuples of each sublist. Identify all tuples with join value c and join them.
 - When a buffer has no more tuple that has not already been considered load another block into this buffer.

32

Yannis Papakonstantinou, CSE232, UCSD

Efficient Sort-Merge Join Example

R	C	RA
1	r ₁	
2	r ₂	
3	r ₃	
...		
20	r ₂₀	

Assume that after first phase of multiway sort we get 4 sublists, 2 for R and 2 for S .

Also assume that each block contains two tuples.

S	C	SA
1	s ₁	
...		
5	s ₅	
...		
16	s ₁₆	
...		
20	s ₂₀	

R	3	7	8	10	11	13	14	16	17	18
1	2	4	5	6	9	12	15	19	20	

S	1	3	5	17	
2	4	16	18	19	20

33

Yannis Papakonstantinou, CSE232, UCSD

Two-Pass Hash-Based Algorithms

- General Idea: Hash the tuples of the input arguments in such a way that all tuples that must be considered together will have hashed to the same hash value.
 - If there are M buffers pick $M-1$ as the number of hash buckets
- Example: Duplicate Elimination
 - Phase 1: Hash each tuple of each input block into one of the $M-1$ bucket/buffers. When a buffer fills save to disk.
 - Phase 2: For each bucket:
 - load the bucket in main memory,
 - treat the bucket as a small relation and eliminate duplicates
 - save the bucket back to disk.
 - **Catch:** Each bucket has to be less than M .
 - **Cost:**

34

Yannis Papakonstantinou, CSE232, UCSD

Hash-Join Algorithms

- Assuming natural join, use a hash function that
 - is the same for both input arguments R and S
 - uses only the join attributes
- Phase 1: Hash each tuple of R into one of the $M-1$ buckets R_i and similar each tuple of S into one of S_i
- Phase 2: For $i=1 \dots M-1$
 - load R_i and S_i in memory
 - join them and save result to disk
- **Question:** What is the maximum size of buckets?
- **Question:** Does hashing maintain sorting?

35

Yannis Papakonstantinou, CSE232, UCSD

Index-Based Join: The Simplest Version

Assume that we do natural join of $R(A,B)$ and $S(B,C)$ and there's an index on S

```

for each Br in R do
  for each tuple r of Br with B value b
    use index of S to find
      tuples {s1, s2, ..., sn} of S with B=b
    output {rs1, rs2, ..., rsn}
  
```

Cost: Assuming R is clustered and non-sorted and the index on S is clustered on B then $B(R)+T(R)B(S)/V(S,B)$ + some more for reading index

Question: What is the cost if R is sorted?

36

Yannis Papakonstantinou, CSE232, UCSD

Opportunities in Joins Using Sorted Indexes

- Do a conventional Sort-Join avoiding the sorting of one or both of the input operands

37

Yannis Papakonstantinou, CSE232, UCSD

Recap

- We have seen:
 - algebraic axioms that can be used in rewriting an algebraic expression (logical query plan) into an equivalent one and possibly more efficient
 - specific algorithms for implementing the various operators
- We will see next
 - how the query processor transforms SQL to logical query plans
 - how the axioms are used to rewrite the logical plan
 - how the appropriate physical plan is picked
 - the last two items require *cost optimization*

38