

Databases 2

Lecture IV

Alessandro Artale

Faculty of Computer Science – Free University of Bolzano

Room: 221

`artale@inf.unibz.it`

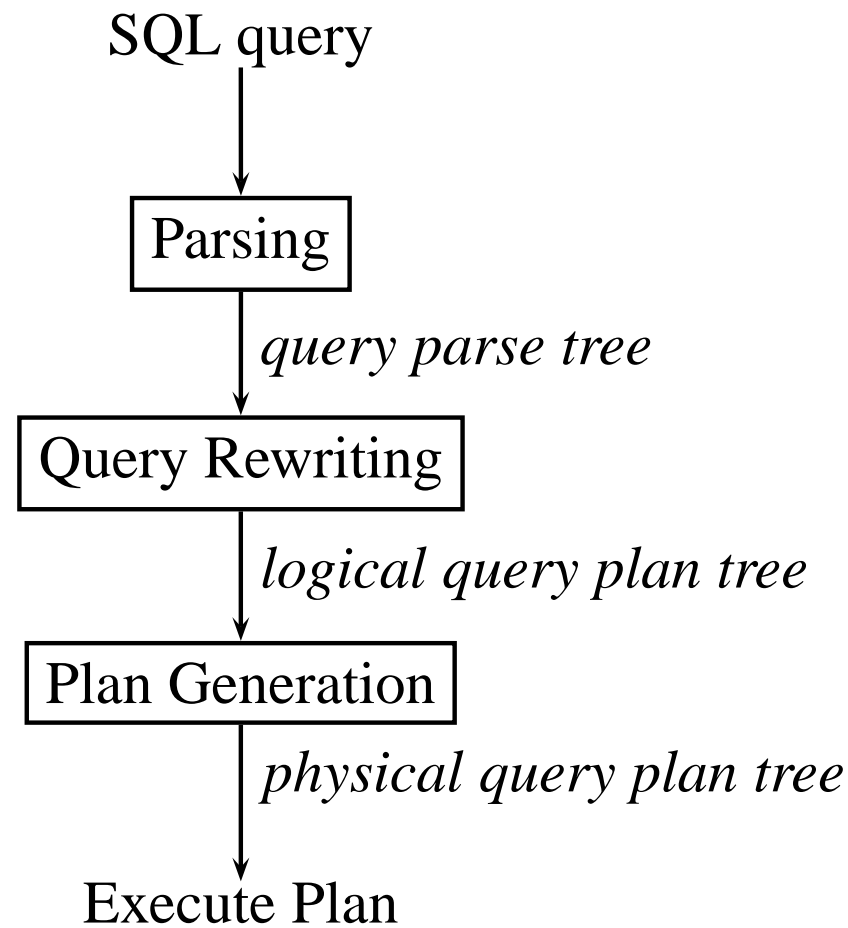
`http://www.inf.unibz.it/~artale/`

2003/2004 – First Semester

Summary of Lecture IV

- **Query Compilation and Optimization**
- Algorithms for Implementing Query Operators
 - One-Pass Algorithms
 - One-And-A-Half Passes Algorithms
 - * Nested-Loop Join
 - Two-Pass Algorithms
 - * Sort-Based Algorithms
 - * Index-Based Algorithms.

Query Compilation Overview



Query Optimization

- Crucial for query optimization are both the *query rewriting* and the *plan generation* phases.
- To select the best query plan decision involve:
 1. Choose the best algorithm for implementing each of the operations of the logical query plan;
 2. Choose the logical query plan that leads to the most efficient algorithm for answering the query.
- Each of these choices depend on:
 1. The size of the relations involved;
 2. Existence of indexes on the relations;
 3. Layout of data on disk.

Cost of Physical-Query-Plan Operators

- **Physical-Query-Plan Operators** implement both relational algebra operators, and generic operations like the *scan* of a relation – to bring into main memory all the tuples of a relation – or the *sort* of a relation.
- The **Cost** associated to the execution of each operator is based on the number of disk I/O – as specified by the **I/O Model of Computation**.
- For estimating the cost of an operator the following parameters are used:
 1. **M**: The number of main memory buffers (assumed equal in size to a disk block) *available* during the execution of an operator;
 2. Parameters that measure size and distribution of data in relations:
 - (a) **B(R)**: number of blocks holding the relation R – if a relation is *clustered*, i.e., stored in fully dedicated blocks, then **B(R)** measures the cost of scanning the relation;
 - (b) **T(R)**: number of tuples in R ;
 - (c) **V(R,A)**: number of distinct values for the attribute A of the relation R .

Cost Parameters: An Example

ACCOUNT

Account#	Branch	Balance
A-101	Downtown	500
A-102	Perryridge	400
A-110	Downtown	600
A-201	Perryridge	900
A-215	Mianus	700
A-217	Brighton	750
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

B(ACCOUNT)	3
T(ACCOUNT)	9
V(ACCOUNT,Account#)	9
V(ACCOUNT,Branch)	6
V(ACCOUNT,Balance)	7

Summary

- Query Compilation and Optimization
- **Algorithms for Implementing Operators**
 - One-Pass Algorithms
 - One-And-A-Half Passes Algorithms
 - * Nested-Loop Join
 - Two-Pass Algorithms
 - * Sort-Based Algorithms
 - * Index-Based Algorithms.

Classification of Operators

Operators can be classified into three classes:

1. **Tuple-at-a-time, unary operators.** Operators that do not require whole relation in main memory – selection, projection.
2. **Full-relation, unary operators.** Operators that require whole relation in main memory – γ, δ .
3. **Full-relation, binary operators.** At least one of the involved relations should reside in main memory – all the remaining operators belong to this class.

Classification of Algorithms

Algorithms for implementing the various Database operations can be classified into three classes:

1. **One-Pass.** Data is read only once from disk;
2. **Two-Pass.** Data is read a first time from disk, processed, written to the disk, and then re-read from disk a second time.
3. **Multipass.** Generalization of the two-pass algorithm to include three or more passes.

Summary

- Query Compilation and Optimization
- Algorithms for Implementing Operators
 - **One-Pass Algorithms**
 - One-And-A-Half Passes Algorithms
 - * Nested-Loop Join
 - Two-Pass Algorithms
 - * Sort-Based Algorithms
 - * Index-Based Algorithms.

One-Pass Algorithms for Tuple-at-Time Operators

Algorithm for *Selection*, $\sigma_C(R)$, and *Projection*, $\pi_A(R)$.

1. Read the blocks of R one at a time into a memory buffer;
2. Perform the desired operation on the current tuples;
3. Move the selected/projected tuples on the output buffer.

Cost Analysis

- The algorithm requires that $M \geq 1$.
- The disk I/O cost is $B(R)$.
- **Note:** The cost can be reduced in the case of a Selection with a condition on an attribute that has an index (see slide 56)

One-Pass Algorithms for Full-Relation, Unary Operators – Duplicate Elimination –

Algorithm for *Duplicate Elimination* $\delta(R)$.

- Read each block of R one at a time, and for every tuple:
 1. If is the first time the tuple is seen: Copy the tuple to the output;
 2. Otherwise: Do not output the tuple.
- One copy of every tuple already seen has to be kept in main memory – this is why we have a Full-Relation algorithm – to make the above decision.

One-Pass Algorithms for Full-Relation, Unary Operators – Duplicate Elimination (cont.) –

- **Problem.**

If we have n tuples, deciding whether a new tuple is among them takes a time proportional to n . The full operation takes time proportional to n^2 .

- **Solution.**

Using appropriate structures to memorize the already seen tuples: A **Balanced Binary Tree** requires $\log n$ to locate a tuple, then the full process takes time proportional to $n \log n$.

Cost Analysis

- The algorithm requires that $M \geq B(\delta(R))$.
- The disk I/O cost is $B(R)$.

One-Pass Algorithms for Full-Relation, Binary Operators – Cartesian Product –

Computing $R \times S$ – assuming that S is smaller in size.

- Read the whole S into main memory (no special structure is needed);
- Read into another buffer one block of R at a time;
- For each tuple $t \in R$ concatenate t with each tuple of S ;
- Output each concatenated tuple.

Cost Analysis

- Number of memory buffers required: $M \geq \min(B(R), B(S)) + 1$.
- The disk I/O cost is $B(R) + B(S)$;
- Considerable amount of *CPU* time to concatenate tuples.

Cartesian Product Algorithm: An Example

STUDENT

StudID#	Name	Course
123	John	CS
142	Marc	CS
154	Mary	Maths
221	Judi	Physics

×

ACCOUNT

Acc#	Branch	Balance
A-101	Downtown	500
A-102	Perryridge	400
A-110	Downtown	600
A-201	Perryridge	900
A-215	Mianus	700
A-217	Brighton	750
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Cartesian Product Algorithm: An Example (Cont.)

- **Cycle 1:** The first block of ACCOUNT is brought in Main Memory.

MAIN MEMORY

STUDENT

123	John	CS
142	Marc	CS
154	Mary	Maths
221	Judi	Physics

ACCOUNT (1st Block)

A-101	Downtown	500
A-102	Perryridge	400
A-110	Downtown	600

OUTPUT

STUDENT \times ACCOUNT(Cycle 1)

StudID#	Name	Course	Acc#	Branch	Balance
123	John	CS	A-101	Downtown	500
123	John	CS	A-102	Perryridge	400
123	John	CS	A-110	Downtown	600
142	Marc	CS	A-101	Downtown	500
142	Marc	CS	A-102	Perryridge	400
142	Marc	CS	A-110	Downtown	600
154	Mary	Maths	A-101	Downtown	500
154	Mary	Maths	A-102	Perryridge	400
154	Mary	Maths	A-110	Downtown	600
221	Judi	Physics	A-101	Downtown	500
221	Judi	Physics	A-102	Perryridge	400
221	Judi	Physics	A-110	Downtown	600

Cartesian Product Algorithm: An Example (Cont.)

- **Cycle 2:** The second block of ACCOUNT is brought in Main Memory.

MAIN MEMORY

STUDENT

123	John	CS
142	Marc	CS
154	Mary	Maths
221	Judi	Physics

ACCOUNT (2nd Block)

A-201	Perryridge	900
A-215	Mianus	700
A-217	Brighton	750

OUTPUT

STUDENT × ACCOUNT (Cycle 2)

StudID#	Name	Course	Acc#	Branch	Balance
123	John	CS	A-201	Perryridge	900
123	John	CS	A-215	Mianus	700
123	John	CS	A-217	Brighton	750
142	Marc	CS	A-201	Perryridge	900
142	Marc	CS	A-215	Mianus	700
142	Marc	CS	A-217	Brighton	750
154	Mary	Maths	A-201	Perryridge	900
154	Mary	Maths	A-215	Mianus	700
154	Mary	Maths	A-217	Brighton	750
221	Judi	Physics	A-201	Perryridge	900
221	Judi	Physics	A-215	Mianus	700
221	Judi	Physics	A-217	Brighton	750

Cartesian Product Algorithm: An Example (Cont.)

- **Cycle 3:** The third block of ACCOUNT is brought in Main Memory.

MAIN MEMORY

STUDENT

123	John	CS
142	Marc	CS
154	Mary	Maths
221	Judi	Physics

ACCOUNT (3rd Block)

A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

OUTPUT

STUDENT × ACCOUNT (Cycle 3)

StudID#	Name	Course	Acc#	Branch	Balan
123	John	CS	A-218	Perryridge	700
123	John	CS	A-222	Redwood	700
123	John	CS	A-305	Round Hill	350
142	Marc	CS	A-218	Perryridge	700
142	Marc	CS	A-222	Redwood	700
142	Marc	CS	A-305	Round Hill	350
154	Mary	Maths	A-218	Perryridge	700
154	Mary	Maths	A-222	Redwood	700
154	Mary	Maths	A-305	Round Hill	350
221	Judi	Physics	A-218	Perryridge	700
221	Judi	Physics	A-222	Redwood	700
221	Judi	Physics	A-305	Round Hill	350

One-Pass Algorithms for Full-Relation, Binary Operators

– Natural Join –

Computing $R(X, Y) \bowtie S(Y, Z)$ – X, Y, Z are set of attributes.

- Read the whole S into a main memory search structure (Balanced Binary Trees), with Y as a search key;
- Read into another buffer one block of R at a time;
- For each tuple $t \in R$ search all the tuples in S that agree with t on the value of Y , using the search structure;
- *Join* such tuples (concatenation with only one repetition of the Y attributes), and then output them.

Cost Analysis

- Number of memory buffers required: $M \geq \min(B(R), B(S)) + 1$.
- The disk I/O cost is $B(R) + B(S)$;
- Considerable amount of *CPU* time to concatenate tuples.

Natural Join Algorithm: An Example

Assume that three tuples fit in one block.

BRANCH

BName	SortCode	Phone
Downtown	156432	01636754
Perryridge	452341	01436754
Mianus	335699	01994545
Brighton	657841	01227676
Redwood	521775	01336546
Round Hill	903215	01764008

ACCOUNT

Acc#	BName	Balance
A-101	Downtown	500
A-102	Perryridge	400
A-110	Downtown	600
A-201	Perryridge	900
A-215	Mianus	700
A-217	Brighton	750
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

⋈

Natural Join Algorithm: An Example (Cont.)

- **Cycle 1:** The first block of ACCOUNT is brought in Main Memory.

MAIN MEMORY

BRANCH

Downtown	156432	01636754
Perryridge	452341	01438300
Mianus	335699	01994545
Brighton	657841	01227676
Redwood	521775	01336546
Round Hill	903215	01764008

ACCOUNT (1st Block)

A-101	Downtown	500
A-102	Perryridge	400
A-110	Downtown	600

OUTPUT

BRANCH ⋈ ACCOUNT (Cycle 1)

BName	SortCode	Phone	Acc#	Balance
Downtown	156432	01636754	A-101	500
Downtown	156432	01636754	A-110	600
Perryridge	452341	01438300	A-102	400

Natural Join Algorithm: An Example (Cont.)

- **Cycle 2:** The second block of ACCOUNT is brought in Main Memory.

MAIN MEMORY

BRANCH

Downtown	156432	01636754
Perryridge	452341	01438300
Mianus	335699	01994545
Brighton	657841	01227676
Redwood	521775	01336546
Round Hill	903215	01764008

ACCOUNT (2nd Block)

A-201	Perryridge	900
A-215	Mianus	700
A-217	Brighton	750

OUTPUT

BRANCH ⋈ ACCOUNT (Cycle 2)

BName	SortCode	Phone	Acc#	Balance
Perryridge	452341	01438300	A-201	900
Mianus	335699	01994545	A-215	700
Brighton	657841	01227676	A-217	750

Natural Join Algorithm: An Example (Cont.)

- **Cycle 3:** The third block of ACCOUNT is brought in Main Memory.

MAIN MEMORY

BRANCH

Downtown	156432	01636754
Perryridge	452341	01438300
Mianus	335699	01994545
Brighton	657841	01227676
Redwood	521775	01336546
Round Hill	903215	01764008

ACCOUNT (3^{rd} Block)

A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

OUTPUT

BRANCH \bowtie ACCOUNT(Cycle 3)

BName	SortCode	Phone	Acc#	Balance
Perryridge	452341	01438300	A-218	700
Redwood	521775	01336546	A-222	700
Round Hill	903215	01764008	A-305	350

Summary

- Query Compilation and Optimization
- Algorithms for Implementing Operators
 - One-Pass Algorithms
 - **One-And-A-Half Passes Algorithms**
 - * **Nested-Loop Join**
 - Two-Pass Algorithms
 - * Sort-Based Algorithms
 - * Index-Based Algorithms.

Nested-Loop Join

- Algorithms with “**One-And-A-Half**”-passes: Used when relations don't fit in main memory (which is the common situation in real Databases).
- One of the arguments is read only once, while the other is read more times.

Nested-Loop Join (cont.)

Computing $R \bowtie S$ using a *tuple-based nested-loop join* algorithm, with S smaller in size – a naive way to implement join.

TUPLE-BASED-NESTED-LOOP-JOIN(R, S)

```
FOR each tuple  $s$  of  $S$  DO
  FOR each tuple  $r$  of  $R$  DO
    IF  $r$  and  $s$  join THEN
      output the resulting tuple;
```

Cost Analysis

- If we buffer each relation one block at a time:

Number of Disk I/O: $B(S) + T(S) \times B(R)$.

- **Example.** Let $B(R) = 1,000$, $B(S) = 500$, $T(S) = 5,000$, then:

ONE-PASS JOIN costs $B(R) + B(S) = 1,500$ disk I/O;

TUPLE-BASED NESTED-LOOP JOIN costs $B(S) + T(S) \times B(R) = 5,000,500$ disk I/O.

Block-Based Nested-Loop Join

Optimize the TUPLE-BASED NESTED-LOOP JOIN algorithm.

Use as much main memory as we can to store S , i.e., if M are the available memory buffers we reserve $M - 1$ buffers to S and 1 to R .

BLOCK-BASED-NESTED-LOOP-JOIN(R, S)

```
FOR each chunk of  $M - 1$  blocks of  $S$  DO
    read them to a main-memory search structure
    with the common attributes as search key;
FOR each tuple  $r$  of  $R$  DO
    search the current tuples of  $S$  that join with  $r$ ;
    output the resulting joining tuples;
```

Block-Based Nested-Loop Join: Cost Analysis

- **Example.** Let $B(R) = 1,000$, $B(S) = 500$, $M = 101$, then:

BLOCK-BASED NESTED-LOOP JOIN. Reading the whole S requires 5 iterations. At each of such iterations we read 100 blocks for S and the whole 1,000 blocks for R – i.e., at each iteration we do 1,100 disk I/O. Then the final cost is **5,500** disk I/O.

TUPLE-BASED NESTED-LOOP JOIN costs $B(S) + T(S) \times B(R) = \mathbf{5,000,500}$ disk I/O.

- **Formal Analysis.**

Number of iterations: $\frac{B(S)}{M-1}$;

Number of blocks per iteration: $M - 1 + B(R)$;

Total Number of Disk I/O: $\frac{B(S)}{M-1} \times (M - 1 + B(R))$;

Good Approximation: $\frac{B(S) \times B(R)}{M}$.

- **Exercise.** What if we start by buffering R instead of S ?

Summary

- Query Compilation and Optimization
- Algorithms for Implementing Operators
 - One-Pass Algorithms
 - One-And-A-Half Passes Algorithms
 - * Nested-Loop Join
 - **Two-Pass Algorithms**
 - * **Sort-Based Algorithms**
 - * Index-Based Algorithms.

Two-Pass Algorithms Based on Sorting

Two- or Multi-Pass Algorithms are used when full relations do not fit in main memory – which is the norm for large databases.

- **General Idea.** Given a relation with $B(R) > M$ then:
 1. Read M blocks of R into main memory;
 2. Sort these M blocks in main memory using efficient structures – for Balanced Binary Trees sorting n elements costs $n \log n$.
 3. Write the sorted buffers into M disk blocks, called *sorted sublist of R* .
Go to step 1 if there are unconsidered blocks for R .
 4. Use a *Second Pass* to “merge” the sorted sublists for computing the final result.
- Steps 1,2,3 above form the common *First Pass* in a Two-Pass algorithm based on sorting.

Two-Pass Algorithms Based on Sorting

– Natural Join –

Computing $R(X, Y) \bowtie S(Y, Z)$ with a sort-based algorithm:

1. “**Sort**” R and S by Y if they are not already sorted;
2. “**Merge**” the sorted relations matching all tuples with common values of Y :
 - (a) Read the current block of both R and S in main memory;
 - (b) Find the tuple with the least value y for Y ;
 - (c) If y does not appear at the front of the other relation then consider the successive least y ;
 - (d) Otherwise, identify **ALL** tuples from **BOTH** relations having sort key y .
Note. *If necessary read subsequent blocks from R and/or S to find **ALL** tuples with search key y . If more than the M available buffers are necessary the algorithm doesn't work anymore.*
 - (e) Output the joining tuples;
 - (f) If either relation has no more unconsidered tuples in main memory, reload the buffer for this relation, and go back to step 2b.

Sort-Based Natural Join: Example

Suppose that three tuples fit on a block, and the sorted relations (w.r.t. **BName**) are:

BRANCH

BName	SortCode	Phone
Brighton	657841	01227676
Downtown	156432	01636754
Mianus	335699	01994545
Perryridge	452341	01436754
Redwood	521775	01336546
Round Hill	903215	01764008

ACCOUNT

Acc#	BName	Balance
A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

⋈

Sort-Based Natural Join: Example (Cont.)

- **Cycle 1:** The first block of both BRANCH and ACCOUNT are brought in Main Memory.

MAIN MEMORY

BRANCH

BName	SortCode	Phone
Brighton	657841	01227676
Downtown	156432	01636754
Mianus	335699	01994545

⋈

ACCOUNT

Acc#	BName	Balance
A-217	Brighton	750
A-101	Downtown	500
A-110	Downtown	600

OUTPUT

BRANCH⋈ACCOUNT(Cycle 1)

BName	SortCode	Phone	Acc#	Balance
Brighton	657841	01227676	A-217	750
Downtown	156432	01636754	A-101	500
Downtown	156432	01636754	A-110	600

Sort-Based Natural Join: Example (Cont.)

- **Cycle 2:** The second block of ACCOUNT is brought in Main Memory.

MAIN MEMORY

BRANCH

BName	SortCode	Phone
Brighton	657841	01227676
Downtown	156432	01636754
Mianus	335699	01994545

⋈

ACCOUNT

Acc#	BName	Balance
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900

OUTPUT

BRANCH⋈ACCOUNT(Cycle 2)

Mianus	335699	01994545	A-215	700
--------	--------	----------	-------	-----

Sort-Based Natural Join: Example (Cont.)

- **Cycle 3:** The second block of BRANCH and the third of ACCOUNT are brought in Main Memory.

MAIN MEMORY

BRANCH

BName	SortCode	Phone
Perryridge	452341	01436754
Redwood	521775	01336546
Round Hill	903215	01764008

⊗

ACCOUNT

Acc#	BName	Balance
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Sort-Based Natural Join: Example (Cont.)

OUTPUT(Cycle 3)

BRANCH ⋈ ACCOUNT(Cycle 3)

Perryridge	452341	01438300	A-102	400
Perryridge	452341	01438300	A-201	900
Perryridge	452341	01438300	A-218	700
Redwood	521775	01336546	A-222	700
Round Hill	903215	01764008	A-305	350

Sort-Based Natural Join: Cost Analysis

- Number of Disk I/O: $5 \times (B(R) + B(S))$.
 1. Sorting with a two-phase merge sort algorithm costs 4 disk I/O per block. Then the sorting phase requires $4 \times (B(R) + B(S))$ disk I/O.
 2. The merge-join phase requires reading each block of the involved relations one time. Thus, this phase costs $(B(R) + B(S))$ disk I/O.
 3. *Total Cost*: $5 \times (B(R) + B(S))$ disk I/O.
- Number of memory buffer required: $M > \sqrt{\max(B(R), B(S))}$.
 - Needed to perform the sort.

Sort-Based Join: Cost Analysis (cont.)

- **Example.** Let $B(R) = 1000$, $B(S) = 500$, $M = 101$, and for no y do the tuples of R and S that agree on y occupy more than 101 blocks, then the cost of different techniques are:

Sort-Based Join: $5 \times (B(R) + B(S)) = 7,500$ disk I/O.

Block-Based Nested-Loop Join: $\frac{B(S)}{M-1} \times (M-1 + B(R)) = 5,500$ disk I/O.

Tuple-Based Nested-Loop Join: $B(S) + T(S) \times B(R) = 5,000,500$ disk I/O.

- **Note.** The cost of BLOCK-BASED NESTED-LOOP JOIN grows following a quadratic function – proportional to $B(S) \times B(R)$ – while SORT-BASED JOIN grows linearly – proportional to $B(R) + B(S)$. Then SORT-BASED JOIN is a better performing algorithm as far as the size of the relations grows.
 - Indeed, try with: $B(R) = 10,000$, $B(S) = 5,000$, $M = 101$.

Two-Pass Algorithms Based on Sort-Merge

- Algorithms still based on sorting.
- Instead of completing the sorting, they combine the second phase of the sorting with the operation implementation itself.

Two-Pass Algorithms Based on Sort-Merge – Duplicate Elimination –

Computing $\delta(R)$ with a sort-merge based algorithm.

First Pass: Generation of the sorted sublists for R (the sorting order is made on an arbitrary attribute).

Second Pass.

1. Read in main memory one block from every sorted sublist;
2. Consider the first tuple from each block and select the least in sorted order, say t ;
3. Output one copy of t ;
4. Remove all copies of t from the blocks currently in memory.
5. If a block is exhausted we read the next block from the same sublist, and if there are tuples t we remove them as well;
6. If there are unconsidered tuples in main memory go back to step 2.

Duplicate Elimination: An Example

Suppose that tuples are integer numbers, only two tuples fit on a block, $M = 3$, and R has 17 tuples:

R								
2,5	2,1	2,2	4,5	4,3	4,2	1,5	2,1	3
1,2	2,2	2,5	2,3	4,4	4,5	1,1	2,3	5
R_1			R_2			R_3		

First Pass.

1. Read the first 3 blocks of R (6 tuples) into the 3 main memory buffers;
2. Sort them and write the sorted sublist R_1 back to the disk;
3. In a similar way generate the sorted sublists R_2, R_3 .

Duplicate Elimination: An Example (cont.)

Second Pass – Cycle 1.

1. Read in main memory the first block from every sorted sublist:

Sublists	Main Memory	Waiting on Disk	
R_1	1, 2	2, 2	2, 5
R_2	2, 3	4, 4	4, 5
R_3	1, 1	2, 3	5

2. The least tuple of the three blocks in main memory is **1**.
3. We copy **1** to the output:

Output Buffer

1				
---	--	--	--	--

Duplicate Elimination: An Example (cont.)

Second Pass – Cycle 1 (cont.).

4. Remove the tuple 1 from all the blocks in memory.

Sublists	Main Memory	Waiting on Disk	
R_1	2	2, 2	2, 5
R_2	2, 3	4, 4	4, 5
R_3		2, 3	5

5. Since the buffer for R_3 is exhausted we read the next block from this sublist in main memory:

Sublists	Main Memory	Waiting on Disk	
R_1	2	2, 2	2, 5
R_2	2, 3	4, 4	4, 5
R_3	2, 3	5	

6. Since there are still tuples in memory go back to step 2.

Duplicate Elimination: An Example (cont.)

Second Pass – Cycle 2.

2. The least tuple of the three blocks in main memory is now 2.
3. We copy 2 to the output:

Output Buffer

1	2			
---	---	--	--	--

4. Remove the tuple 2 from all the blocks in memory:

Sublists	Main Memory	Waiting on Disk	
R_1		2, 2	2, 5
R_2	3	4, 4	4, 5
R_3	3	5	

Duplicate Elimination: An Example (cont.)

Second Pass – Cycle 2 (cont.).

5. Since the buffer for R_1 is exhausted we read the next block from this sublist, and since there are tuples **2** we remove them as well;

Sublists	Main Memory	Waiting on Disk
R_1	5	
R_2	3	4, 4 4, 5
R_3	3	5

6. Since there are still tuples in memory go back to step 2.

Duplicate Elimination: An Example (cont.)

Second Pass – End.

To complete the process, the tuples 3, 4, 5 are consumed in succession giving the desired output:

Output Buffer

1	2	3	4	5
---	---	---	---	---

Duplicate Elimination: Cost Analysis

- Number of Disk I/O: $3 \times B(R)$.
 1. $B(R)$ to read each block of R when creating the sorted sublists;
 2. $B(R)$ to write each sublist back to disk;
 3. $B(R)$ to read each block from the sublist in the second pass.
- Number of memory buffer required: $M \geq \sqrt{B(R)}$.
 1. Assuming M buffers are available. Then, each sorted sublist can be at most M blocks long;
 2. For the second pass one block for *EACH* sublist should be in main memory. Then, the number of sublists is at most M ;
 3. Given M available buffers, we can process at most M sublists, each at most M blocks long: $B(R) \leq M^2$.
- **Exercise.** Try the example with $M = 2$ and verify that the algorithm goes in *Memory Overflow*.

Sort-Merge Natural Join

Computing $R(X, Y) \bowtie S(Y, Z)$ with a sort-merge based algorithm.

First Pass: Create sorted sublists for both R and S of size M using Y as sort key.

Second Pass.

1. Bring the first block of **EACH** sublist into main memory;
2. Find the tuple with the least value y for Y ;
3. If y does not appear in the blocks of the other relation then consider the successive least y ;
4. Otherwise, identify **ALL** the tuples from **BOTH** relations having sort key y —perhaps bringing new blocks in main memory;
5. Output the joining tuples;
6. If the buffer for one of the sublists is exhausted reload the buffer for this sublist.
7. If there are unconsidered tuples in main memory go back to step 2.

Sort-Merge Natural Join: An Example

- Assume that the joining attribute is of type integer, only two tuples fit on a block, $M = 5$, and R and S both have 17 tuples—in the following we represent only the joining attribute.
- First Pass:** generate the sorted sublists:

R								
2,5	2,1	2,2	4,5	4,3	4,2	1,5	2,1	3
1,2	2,2	2,3	4,4	5,5	1,1	2,2	3,4	5
R_1					R_2			
S								
2,1	2,1	1,2	2,3	3,3	3,2	6,5	4,4	4
1,1	1,2	2,2	2,3	3,3	2,3	4,4	4,5	6
S_1					S_2			

Sort-Merge Natural Join: An Example (Cont.)

Second Pass – Cycle 1. Note: We show the situation in main memory for each value of the joining attribute.

- Since after Steps 1-3, **1** is the least **common** value for the joining attribute we read in main memory **ALL** the blocks for this joining attribute (as specified in Step 4):

Sublists	Main Memory	Waiting on Disk
R_1	1, 2	2, 2 2, 3 4, 4 5, 5
R_2	1, 1	2, 2 3, 4 5
S_1	1, 1 1, 2	2, 2 2, 3 3, 3
S_2	2, 3	4, 4 4, 5 6

Note. For the Sublist S_1 two blocks are read in main memory respecting the Step 4 of the algorithm.

- (Step 5) The Natural Join between the tuples having joining attribute = 1 is computed and moved to the output.

Sort-Merge Natural Join: An Example (Cont.)

- (Step 6) The buffer for R_2 is exhausted: we read its successive block.

Sublists	Main Memory	Waiting on Disk			
R_1	1, 2	2, 2	2, 3	4, 4	5, 5
R_2	2, 2	3, 4	5		
S_1	1, 2	2, 2	2, 3	3, 3	
S_2	2, 3	4, 4	4, 5	6	

Sort-Merge Natural Join: An Example (Cont.)

Second Pass – Cycle 2.

- (Steps 2-4) Read in main memory **ALL** the blocks for the joining attribute 2:

Sublists	Main Memory			Waiting on Disk		
R_1	1, 2	2, 2	2, 3	4, 4	5, 5	
R_2	2, 2			3, 4	5	
S_1	1, 2	2, 2	2, 3	3, 3		
S_2	2, 3			4, 4	4, 5	6

Since there are only 5 buffers available ($M = 5$) while this step requires 8 buffers the algorithm fails with a memory overflow error.

Sort-Merge Natural Join: Cost Analysis

- Number of Disk I/O: $3 \times (B(R) + B(S))$.
 1. Creating the sorted sublist costs 2 disk I/O per each block of data;
 2. The merge-join phase requires reading each block of the involved relations one more time. Thus the merge-join phase costs $(B(R) + B(S))$ disk I/O.
 3. *Total Cost*: $3 \times (B(R) + B(S))$ disk I/O.
- Number of memory buffer required: $M \geq \sqrt{B(R) + B(S)}$.
 1. This is obtained similarly to the case of sort-based duplicate elimination noting that now $B(R) + B(S)$ blocks need to be processed:
 $M^2 \geq B(R) + B(S)$.

Sort-Merge Natural Join: Cost Analysis. Example

- **Example.** Let $B(R) = 1,000$, $B(S) = 500$, $M = 101$, and for no y do the tuples of R and S that agree on y occupy more than 101 blocks. Then, the cost of different techniques are:

Sort-Merge Join: $3 \times (B(R) + B(S)) = 4,500$ disk I/O.

Sort-Based Join: $5 \times (B(R) + B(S)) = 7,500$ disk I/O.

Block-Based Nested-Loop Join: $\frac{B(S)}{M-1} \times (M-1 + B(R)) = 5,500$ disk I/O.

Tuple-Based Nested-Loop Join: $B(S) + T(S) \times B(R) = 5,000,500$ disk I/O.

Summary

- Query Compilation and Optimization
- Algorithms for Implementing Operators
 - One-Pass Algorithms
 - One-And-A-Half Passes Algorithms
 - * Nested-Loop Join
 - **Two-Pass Algorithms**
 - * Sort-Based Algorithms
 - * **Index-Based Algorithms**

Index-Based Algorithms

- Index-Based Algorithms are useful for the Selection operator, but also the other operators can take advantage.
- In the following we describe how Selection and Join operators can be implemented taking into account the existence of indexes.

Index-Based Selection

- The One-Pass algorithm scan **ALL** the tuples: It costs $B(R)$ disk I/O.
- Suppose the query is $\sigma_{A=v}(R)$, and there is an index on the attribute A .
- **Algorithm:** Search the index for the value v and get the pointers to **EXACTLY** those tuples of R having value v for A .

Index-Based Selection: Cost Analysis

- $V(R,A)$: number of distinct values for the attribute A of the relation R .
- Number of Disk I/O
 - If the index is a *secondary index* – tuples are spread on different blocks – then $\sigma_{A=v}(R)$ requires about $\left\lceil \frac{T(R)}{V(R,A)} \right\rceil$ Disk I/O.
 - If the index is a *primary index* – tuples with a given value are stored one next to the other – then $\sigma_{A=v}(R)$ requires *at least* $\left\lceil \frac{B(R)}{V(R,A)} \right\rceil$ (if values are uniformly distributed).
- **Note:** If the index does not fit in main memory we need to consider the disk I/O for index lookup.

Index-Based Selection: An Example

The number $\left\lceil \frac{T(R)}{V(R,A)} \right\rceil$ is an estimate of the number of tuples with a fixed value for the attribute A .

ACCOUNT

Account#	Branch	Balance
A-101	Downtown	500
A-102	Downtown	400
A-110	Downtown	600
A-201	Downtown	900
A-215	Downtown	400
A-217	Downtown	500
A-218	Downtown	900
A-222	Downtown	600

$T(\text{ACCOUNT})$	8
$V(\text{ACCOUNT}, \text{Account\#})$	8
$V(\text{ACCOUNT}, \text{Branch})$	1
$V(\text{ACCOUNT}, \text{Balance})$	4
$\left\lceil \frac{T(\text{ACCOUNT})}{V(\text{ACCOUNT}, \text{Account\#})} \right\rceil$	1
$\left\lceil \frac{T(\text{ACCOUNT})}{V(\text{ACCOUNT}, \text{Branch})} \right\rceil$	8
$\left\lceil \frac{T(\text{ACCOUNT})}{V(\text{ACCOUNT}, \text{Balance})} \right\rceil$	2

Selection: An Example

$$\sigma_{\text{Branch-Name}=\text{"Stretford"} \wedge \text{Customer-Name}=\text{"John"}}(\text{DEPOSIT})$$

Suppose that $T(\text{Deposit}) = 10,000$, $B(\text{Deposit}) = 500$ (i.e., 20 tuples per block),
 $V(\text{Deposit}, \text{Branch-Name}) = 50$, $V(\text{Deposit}, \text{Customer-Name}) = 200$.

We estimate the cost of the query assuming that the following indexes exist:

1. A primary index for Branch-Name;
 2. A secondary index for Customer-Name.
- Using the primary index we need to read $\frac{T(\text{Deposit})}{V(\text{Deposit}, \text{Branch-Name})} = 200$ tuples, and $\frac{B(\text{Deposit})}{V(\text{Deposit}, \text{Branch-Name})} = \frac{500}{50} = 10$ blocks.
 - Using the secondary index we need to read $\frac{T(\text{Deposit})}{V(\text{Deposit}, \text{Customer-Name})} = 50$ tuples. But since tuples are not stored in sequence, the number of blocks needed is *at most* 50.
 - ONE-PASS ALGORITHM takes $B(\text{Deposit}) = 500$ disk I/O.

Selection: An Example (cont.)

- Intersecting pointers from both indexes in main memory: $P_1 \cap P_2$.
- One tuple in:

$$V(\text{Deposit, Branch-Name}) \times V(\text{Deposit, Customer-Name}) = 50 \times 200 = 10,000$$

has both Branch-Name = *Stretford* and Customer-Name = *John*. Then, $P_1 \cap P_2 = 1$, and we need to access just **1** block.

Summing up.

1. No Index: **500** disk I/O;
2. Primary Index: **10** disk I/O;
3. Secondary Index: **50** disk I/O;
4. Intersecting Pointers: **1** disk I/O;
5. Index with *Concatenated Keys*: **1** disk I/O.

What if there is a *Disjunctive Selection*?

Index-Based Join

Computing $R(X, Y) \bowtie S(Y, Z)$ assuming that S has an index for Y :

1. Read each block of R and for each tuple r , with r_Y its projection on the Y attributes:
 - (a) Use the index on S with search key r_Y to find all the tuples of S that can be joined with r ;
 - (b) Output the joining tuples.

Note: This procedure is essentially a *nested loop* algorithm that just takes advantage from the existence of an index to *selectively* search S .

Index-Based Join: Cost Analysis

Cost Analysis

- Reading the full R costs $B(R)$ disk I/O.
- For each tuple of R we must read about $\frac{T(S)}{V(S,Y)}$ tuples of S , then we need to distinguish two cases:
 1. S has a secondary index for Y then reading S costs
 $T(R) \times \left\lceil \frac{T(S)}{V(S,Y)} \right\rceil \rightarrow$ Total Number of disk I/O: $B(R) + T(R) \times \left\lceil \frac{T(S)}{V(S,Y)} \right\rceil$;
 2. S has a primary index for Y then reading S costs
 $T(R) \times \left\lceil \frac{B(S)}{V(S,Y)} \right\rceil \rightarrow$ Total Number of disk I/O: $B(R) + T(R) \times \left\lceil \frac{B(S)}{V(S,Y)} \right\rceil$.

Index-Based Join: An Example

Consider our running example:

$M = 101$, $B(R) = 1,000$, $B(S) = 500$, $T(R) = 10,000$, $T(S) = 5,000$, $V(S, Y) = 100$ (i.e., $\frac{T(S)}{V(S, Y)} = 50$), then:

1. S has a secondary index for Y :

$$\text{Join Cost: } B(R) + T(R) \times \left\lceil \frac{T(S)}{V(S, Y)} \right\rceil = 1,000 + 10,000 \times \frac{5,000}{100} =$$

501,000 disk I/O;

2. S has a primary index for Y :

$$\text{Join Cost: } B(R) + T(R) \times \left\lceil \frac{B(S)}{V(S, Y)} \right\rceil = 1,000 + 10,000 \times \frac{500}{100} =$$

51,000 disk I/O.

3. TUPLE-BASED NESTED-LOOP JOIN: $B(S) + T(S) \times B(R) =$
5,000,500 disk I/O.

BLOCK-BASED NESTED-LOOP JOIN: $\frac{B(S)}{M-1} \times (M-1 + B(R)) =$
5,500 disk I/O.

SORT-MERGE JOIN costs $3 \times (B(R) + B(S)) =$ **4,500** disk I/O.

Index-Based Join: An Example (cont.)

A better situation if there is an index on the bigger relation:

- R has a primary index for Y , and $\frac{T(R)}{V(R,Y)} = 50$ (i.e., $V(R,Y) = 200$):

$$\text{Join Cost} = B(S) + T(S) \times \left\lceil \frac{B(R)}{V(R,Y)} \right\rceil = 500 + 5,000 \times \frac{1000}{200} =$$

25,500 disk I/O.

- If Y is also a key, i.e., $\frac{T(R)}{V(R,Y)} = 1$ (i.e., $V(R,Y) = 10,000$):

$$\text{Join Cost} = B(S) + T(S) \times 1 = 500 + 5,000 = \mathbf{5,500}$$
 disk I/O.

Summary of Join Algorithms Performances

- *Sort-Based Join* algorithms are in general the most efficient algorithms to compute a Join;
- *Block-Based Nested-Loop Join* algorithms are used when there are many tuples sharing a common attribute (sort-based join could stop because of a memory overflow);
- *Index-Based Join* algorithms are comparable with sort-based join when the biggest relation has a primary index on the joining attribute.

Summary of Lecture IV

- Query Compilation and Optimization
- Algorithms for Implementing Operators
 - One-Pass Algorithms
 - One-And-A-Half Passes Algorithms
 - * Nested-Loop Join
 - Two-Pass Algorithms
 - * Sort-Based Algorithms
 - * Index-Based Algorithms