Query Optimization

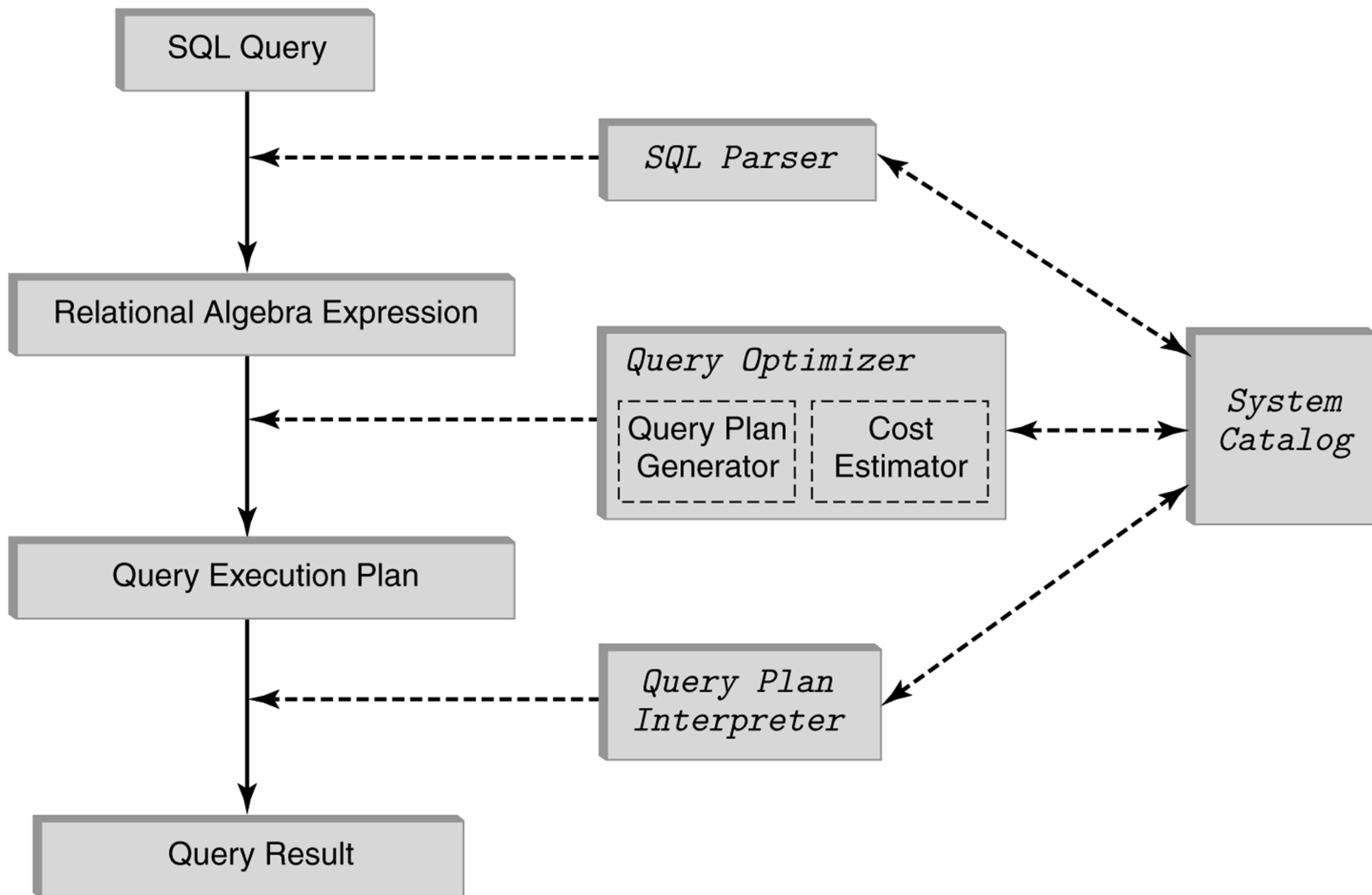
Cost-Based

SWEN304/SWEN435

Lecturer: Dr Hui Ma

Engineering and Computer Science

Outline – Query Evaluation in DBMS



Victoria University of Wellington, Te Whare Wānanga o te Ūpoko o te Ika a Māui

Outline – Cost-based Query Optimization

- Why cost-based optimization?
- How to measure cost of query operations?
  - Cost function of a projection, selection, join, …
- How to evaluate query operations?
  - Algorithms for projection, selection, join, …
- Query tree of physical operators
    - Reading: chapters 17, 18, 19 of 6/E of the textbook
    - Supposed knowledge File Organization

Example

R (Reserves)

sid	bid	day
58	101	10/09/13
22	103	11/09/13
31	103	11/09/13

S (Sailor)

sid	sname	rating	age
22	Jerry	7	25
31	Tom	8	30
58	Minny	10	22

B (Boats)

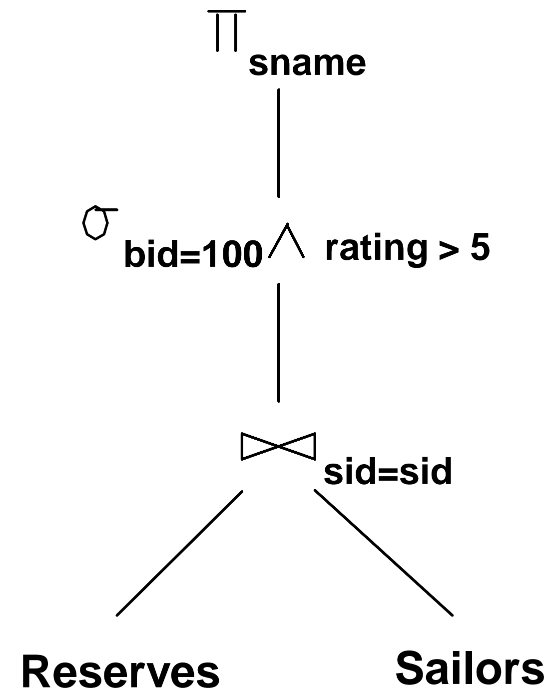
bid	bname	color
101	Dragon	blue
102	Moon	red
103	Star	green
104	Clipper	blue
105	Mary	green

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
  R.bid=100 AND S.rating>5
```

Example

- Here is a query tree for the SQL query
  - Is this optimal?
  - How good is it?
  - How to measure “goodness”?
  - Need to estimate its costs!

```
SELECT S.sname
FROM Reserves R, Sailors S
WHERE R.sid=S.sid AND
  R.bid=100 AND S.rating>5
```



Cost-Based Optimization

- A good query optimizer does not rely solely on heuristic rules
- It chooses a query execution plan which has the lowest cost estimate
  - Or at least one whose costs are reasonably good
- After heuristic rules are applied to a query, there still remain a number of alternative ways to execute it
  - These alternative ways relay on the existence of different auxiliary data structures and algorithms
- Query optimizer estimates the cost of executing each of alternative ways, and chooses the one with the lowest cost

Cost Components of a Query Execution

- Secondary storage access cost:
  - Reading data blocks during data searching,
  - Writing data blocks on disk, and
  - Storage cost (cost of storing intermediate files)
- Computation cost (CPU cost)
- Main memory cost (buffer cost)
- Communication cost
- Very often, only secondary storage access cost is considered
- So, the cost C will be the data size of disk accesses

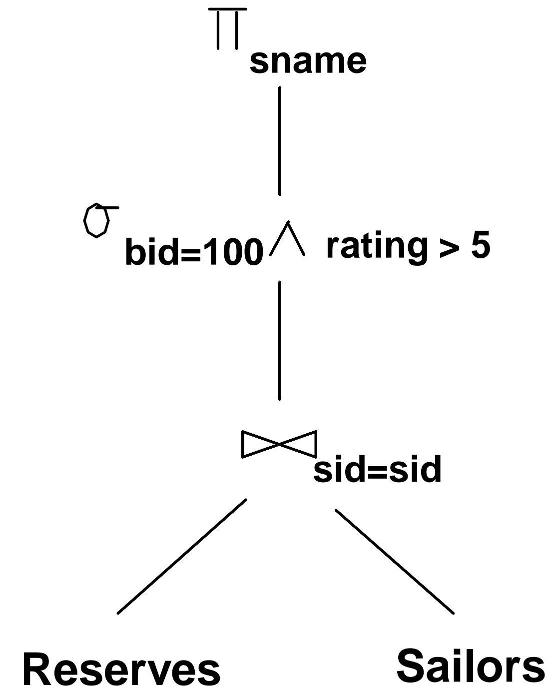
Victoria University of Wellington — Te Whare Wānanga o te Ūpoko o te Ika a Māui

Some Assumptions

- To make it simpler, we shall suppose that:
  - All tuple fields are of a fixed size (although variable field size tuples are very frequent in practice)
  - All the intermediate query results are **materialized** (although there are some advanced optimizers that apply **pipelined** approach)
    - **Materialized:** intermediate query results are stored on a disk as temporary relations
    - **Pipelining**: tuples of the intermediate results are subjected to all subsequent operations without temporary storing

Example

- Materialized evaluation:
  - evaluate one operation at a time, starting from the bottom
  - intermediate results are materialized into temporary relations (write to disk)
  - E.g., the result of the join is materialized, the temporary relation is then read from disk to compute the selection
  - Similarly, the result of the selection is materialized, and then from disk to compute the projection



Victoria University of Wellington — Te Whare Wānanga o te Ūpoko o te Ika a Māui

How to estimate Query Computation Costs?

Recall from our exercises:

Student

LName	FName	StudId	Major
Smith	Susan	131313	Comp
Bond	James	007007	Math
Smith	Susan	555555	Comp
Cecil	John	010101	Math

Course

PName	CourId	Points	Dept
DB Sys	C302	15	Comp
SofEng	C301	15	Comp
DisMat	M214	22	Math
Pr&Sys	C201	22	Comp

Grades

StudId	CourId	Grade
007007	C302	A+
555555	C302	ω
007007	C301	A
007007	M214	A+
131313	C201	B-
555555	C201	C
131313	C302	ω
007007	C201	A
010101	C201	ω

6. Find students who major in 'Math' and got 'A+' in at least one course offered by computer science department

Victoria University of Wellington — Te Whare Wānanga o te Ūpoko o te Ika a Māui

Heuristic Query Optimization: An Example

Recall our query answer:

- **SQL query**:

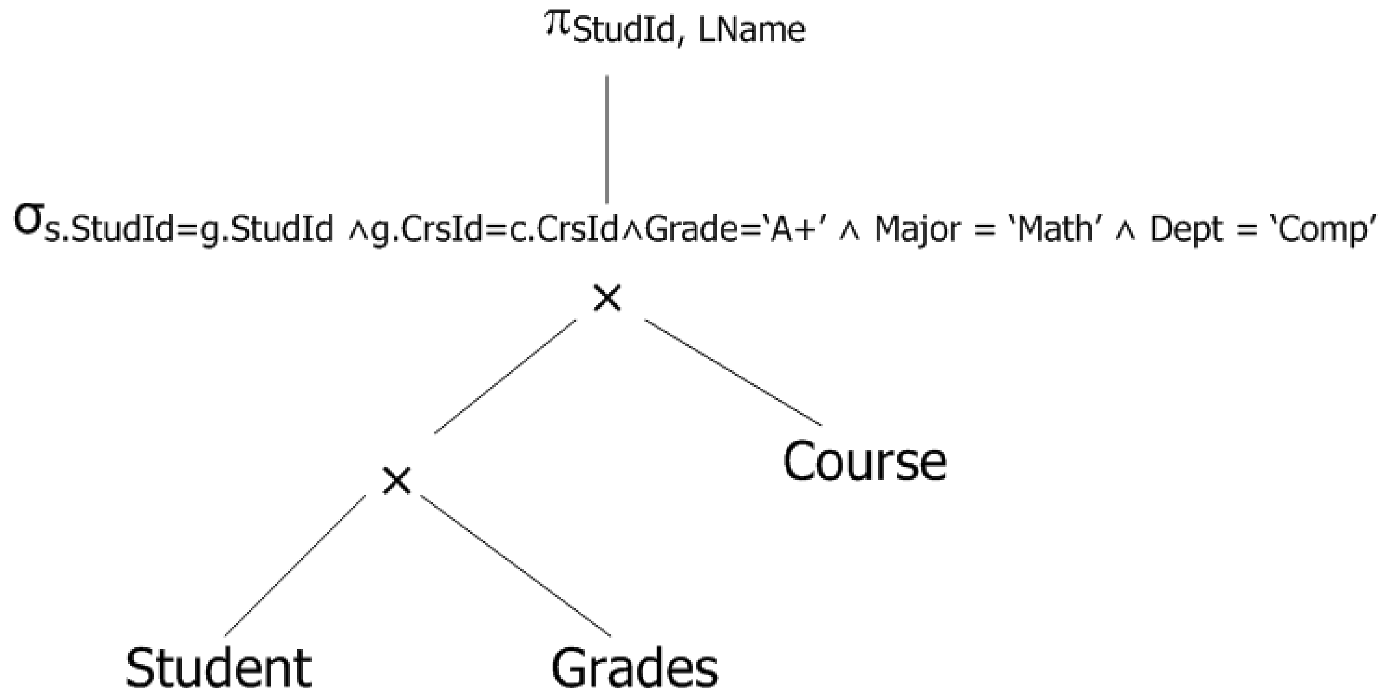
```
SELECT StudId, Lname
FROM Student s, Grades g, Course c
WHERE s.StudId=g.StudId AND g.CrsId= c.CrsId
AND Grade = 'A+' AND Dept = 'Comp' AND Major = 'Math';
```

- Transferred into **Relational Algebra**:

$$\pi_{StudId,\ LName}(\sigma_{s.StudId=g.StudId\ \wedge\ g.CrsId=c.CrsId\ \wedge\ Grade='A+'\ \wedge\ Major='Math'\ \wedge\ Dept='Comp'}(Course \times (Student \times Grades)))$$

How to estimate Query Computation Costs?

- Recall our initial query tree:



  - We start with the sizes of the relations, and then work bottom-up.

How to estimate Query Computation Costs?

- We start with the sizes of the relations:
  - Estimate the size for each attribute,
  - then sum up to get the size of a tuple
  - Then multiply with the number of tuples in a relation

  - We apply this approach to the Student table and obtain:
  - One student tuple requires 23B + 23B + 8B + 11B = 65B
  - The entire Student table requires 4 x 65B = 260B

Student

LName	FName	StudId	Major
Smith	Susan	131313	Comp
Bond	James	007007	Math
Smith	Susan	555555	Comp
Cecil	John	010101	Math
23B	**23B**	**8B**	**11B**

Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga
o te Ūpoko o te Ika a Māui

How to estimate Query Computation Costs?

- We start with the sizes of the relations:
  - For the Course table we obtain: 4 x 34B = 136B
  - For the Grades table we obtain: 9 x 16B = 144B

Student

LName	FName	StudId	Major
Smith	Susan	131313	Comp
Bond	James	007007	Math
Smith	Susan	555555	Comp
Cecil	John	010101	Math
23B	23B	8B	11B

Course

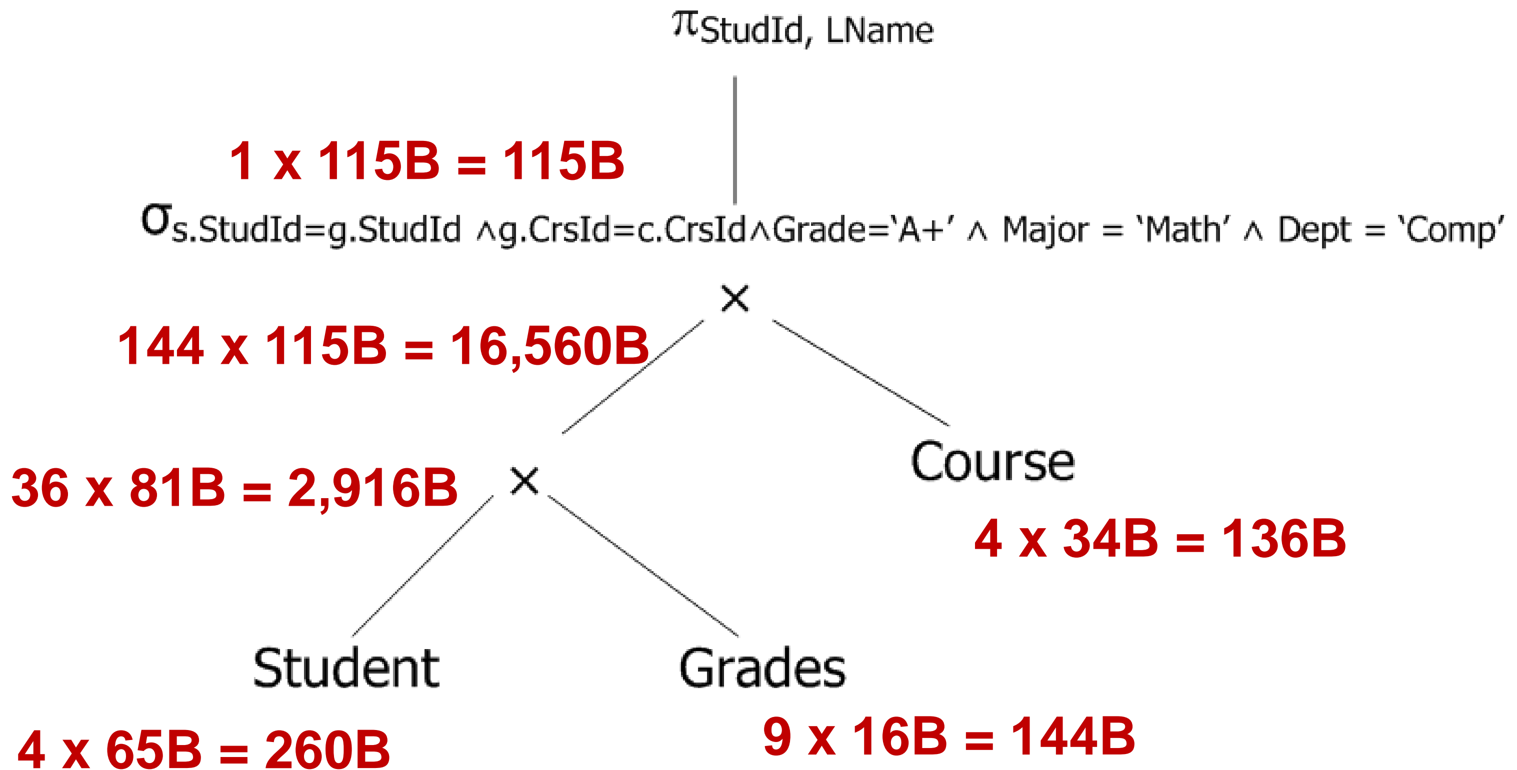
PName	CourId	Points	Dept
DB Sys	C302	15	Comp
SofEng	C301	15	Comp
DisMat	M214	22	Math
Pr&Sys	C201	22	Comp
15B	5B	2B	11B

Grades

StudId	CourId	Grade
007007	C302	A+
555555	C302	ω
007007	C301	A
007007	M214	A+
131313	C201	B-
555555	C201	C
131313	C302	ω
007007	C201	A
010101	C201	ω
8B	5B	3B

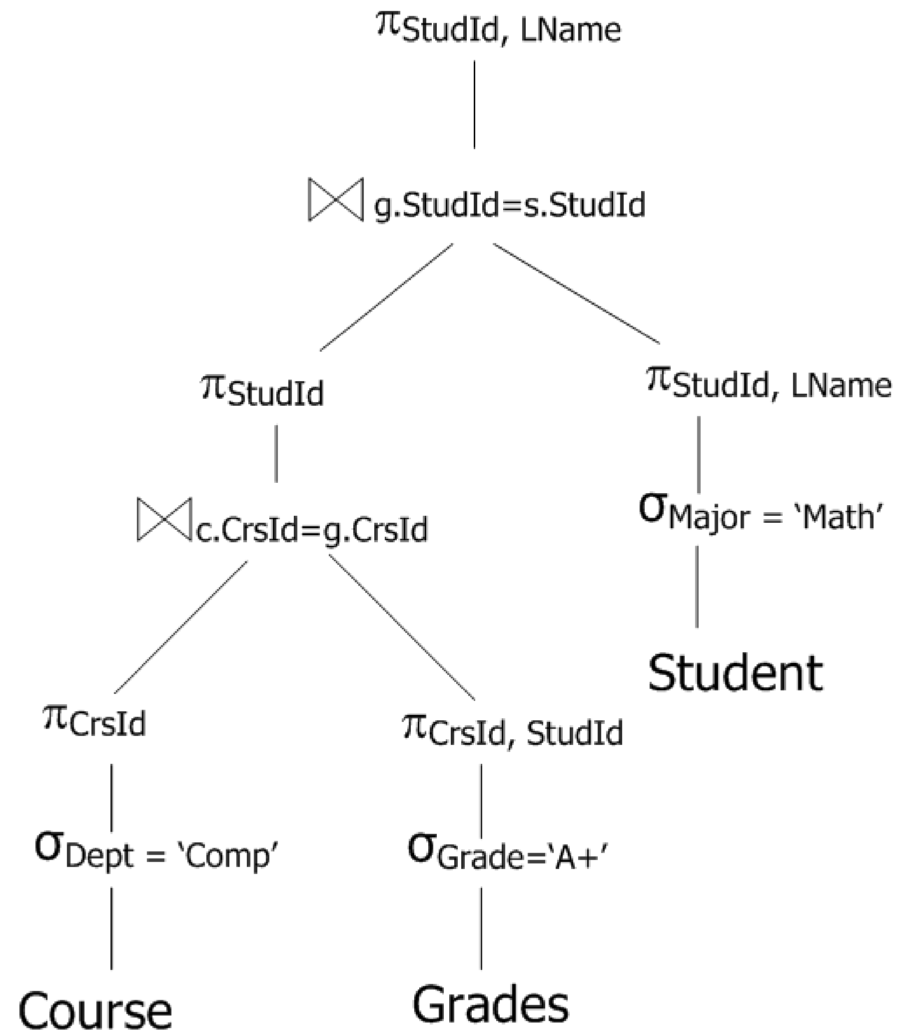
How to estimate Query Computation Costs?

- We start with the sizes of the relations, and then work bottom-up.
  - For each intermediate result we estimate the size based on the inputs.

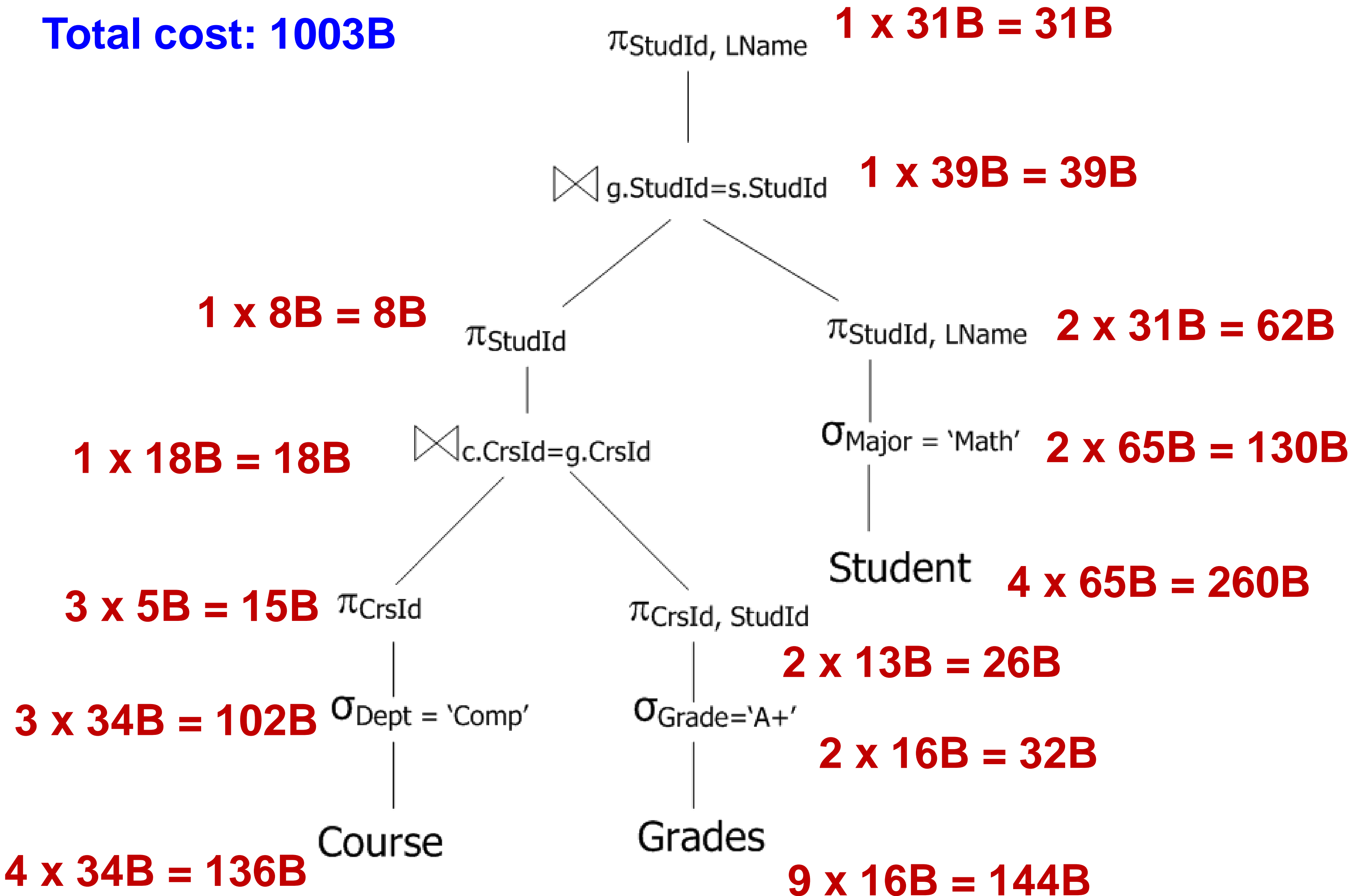


How to estimate Query Computation Costs?

- Recall our optimized query tree:
  - We estimate the query costs for this tree, too.
  - We start with the sizes of the relations, and then work bottom-up.



How to estimate Query Computation Costs?



Which Query Tree is Preferable?

- Note: Both query trees compute the same query result, therefore the size of the result is the same

- When intermediate results do not fit into the main memory, then they have to be stored on hard disk
  - Hard disk access is expensive (= time-consuming)
  - Therefore we want to avoid them
  - Therefore we prefer query trees where the total size of intermediate results is as small as possible
  - In our example:
    - The initial query tree has intermediate sizes up to 20,162B
    - The optimized query tree has much smaller intermediate sizes (1003B)
    - Therefore we prefer the optimized query tree

Query Computation Costs for Unary Operators

- selection σ_C is linear in the size number n of tuples of the involved relation
  - scan the relation one tuple after the other
  - check for each tuple, whether the condition C is satisfied or not
  - keep exactly those tuples satisfying C
- projection π_{AL} is in $O(n \cdot \log n)$ with the number n of tuples
  - order the relation according to the attributes in AL
    (this is the most costly part leading to the complexity in $O(n \cdot \log n)$)
  - scan the relation one tuple after the other
  - project each tuple to the attributes in AL and check, whether result is the same as for previous tuple (duplicate elimination)
  - Note: SQL does not eliminate tuples, i.e. costs of projection are in $O(n)$,
    but `DISTINCT` needs the ordering
- renaming ρ_f can be neglected

Query Computation Costs for Binary Operators

- join $\bowtie$ is in $O(n \cdot \log n)$ with $n = n_1 + n_2$, where n_i are the respective numbers of tuples in the two relations involved
  - the easiest idea is to use a nested loop:
    - scan the first relation one tuple after the other
    - for each tuple scan the second relation to find matching tuples, i.e., those coinciding with the given tuple on the common attributes
    - in case tuples match, take the joined tuple into the result relation
  - more efficient is the merge join:
    - sort both relations (this is the most costly part)
    - scan both relations simultaneously to find matching tuples
    - in case tuples match, take the joined tuple into the result relation
- union $\cup$ is in $O(n \cdot \log n)$ with $n = n_1 + n_2$, where n_i are the respective numbers of tuples in the two relations involved (analogously for difference $-$)
  - sort both relations as for the merge join
  - scan simultaneously to detect duplicates

Victoria University of Wellington — Te Whare Wānanga o te Ūpoko o te Ika a Māui

Estimating the Size of Relations

- let $R = \{A_1, \dots, A_k\}$ be a relation schema
- determine the size of a relation r over R:
  - let n denote the average number of tuples in the relation r
  - let ℓ_j denote the the average space (e.g., in bits) for attribute A_j in a tuple in r
  - then $n \cdot \sum_{j=1}^{k} \ell_j$ is the space needed for the relation r
- determine the size of intermediate relations in a query using the query tree:
  - assign the size of the relation to each leaf node R
  - for a renaming node the assigned size is exactly the size s assigned to the successor

Estimating the Size of Intermediate Results

- for a selection node σ_C the assigned size is $a_C \cdot s$, where s is the size assigned to the successor and $100 \cdot a_C$ is the average percentage of tuples satisfying C

- for a projection node π_{R_i} the assigned size is $(1 - C_i) \cdot s \cdot \frac{r_i}{r}$, where r_i (r) is the average size of a tuple in a relation over R_i (R), s is the size assigned to the successor and C_i is the probability that two tuples coincide on R_i

  $(1 - C_i) \cdot s \cdot \frac{r_i}{r} = (1 - C_i) \cdot n \cdot r_i$ where n is average number of tuples in R-relation

- for a join node the assigned size is $\frac{s_1}{r_1} \cdot p \cdot \frac{s_2}{r_2} \cdot (r_1 + r_2 - r)$, where s_i are the sizes of the successors, r_i are the corresponding tuple sizes, r is the size of a tuple over the common attributes and p is the matching probability

Natural join needs to remove duplicate attributes
For equi-join, r = 0

- for a union node the assigned size is $s_1 + s_2 - p \cdot s_1$ with the probability p for tuple of R_1 to coincide with a tuple over R_2

- for a difference node the assigned size is $s_1 \cdot (1 - p)$ where $(1 - p)$ is probability that tuple from R_1-relation does not occur as tuple in R_2-relation

Victoria University of Wellington, Te Whare Wānanga o te Ūpoko o te Ika a Māui

Estimating the Size of Intermediate Results

Recall from our exercises:

Student

LName	FName	StudId	Major
Smith	Susan	131313	Comp
Bond	James	007007	Math
Smith	Susan	555555	Comp
Cecil	John	010101	Math

Course

PName	CourId	Points	Dept
DB Sys	C302	15	Comp
SofEng	C301	15	Comp
DisMat	M214	22	Math
Pr&Sys	C201	22	Comp

Grades

StudId	CourId	Grade
007007	C302	A+
555555	C302	ω
007007	C301	A
007007	M214	A+
131313	C201	B-
555555	C201	C
131313	C302	ω
007007	C201	A
010101	C201	ω

Find students who major in ‘Math’ and got ‘A+’ in at least one course offered by computer science department

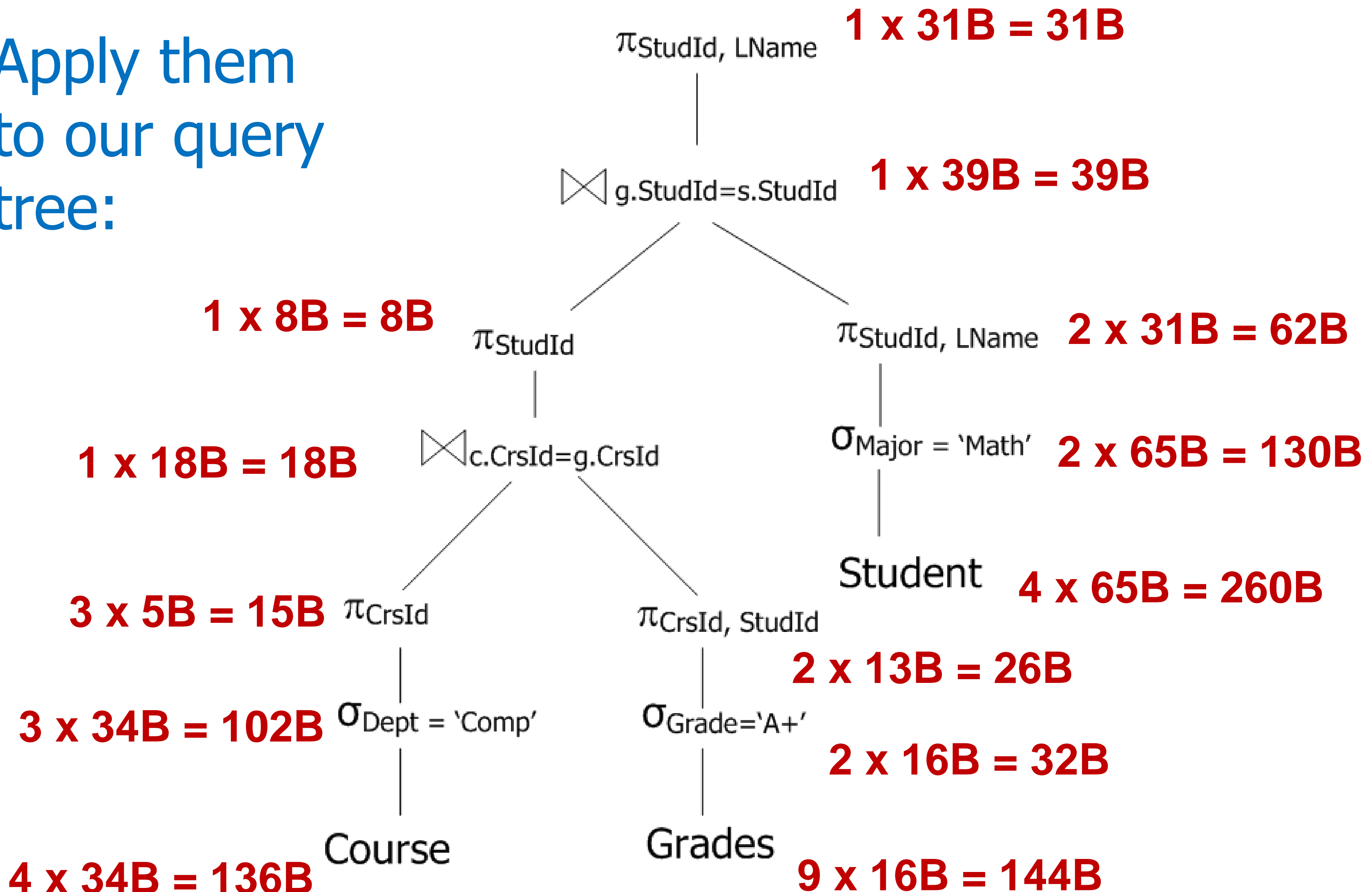
Estimating the Size of Intermediate Results

- Example:
  - We inspect the unary and binary operators that are used and estimate the parameters for them:
    - $\sigma_{Major = 'Math'}$ with selection rate $\alpha_C = 0.5$
    - $\sigma_{Dept = 'Comp'}$ with selection rate $\alpha_C = 0.75$
    - $\sigma_{Grade='A+'}$ with selection rate $\alpha_C = 0.22$

    - π_{StudId} with coincidence rate C = 0.0
    - $\pi_{CrsId,StudId}$ with coincidence rate C = 0.0
    - π_{StudId} with coincidence rate C = 0.0
    - $\pi_{StudId,LName}$ with coincidence rate C = 0.0

Estimating the Size of Intermediate Results

- Apply them to our query tree:



Summary

- DBMS processes a declarative query by converting it to the query tree of logical operators, and by optimizing it
  - looks for a reasonably efficient strategy to implement a query
- Heuristic optimization and cost based optimization are two basic optimization techniques
- Cost based optimization is applied to the result of heuristic optimization
  - exhaustive analyze the number of disk accesses of alternative available methods and algorithms to execute a query
- Techniques that improve query cost:
  - Indexing,
  - Using larger memory,
  - Sorting