

Query Optimization

Heuristic Optimization

SWEN304/SWEN435

Lecturer: Dr Hui Ma

Engineering and Computer Science

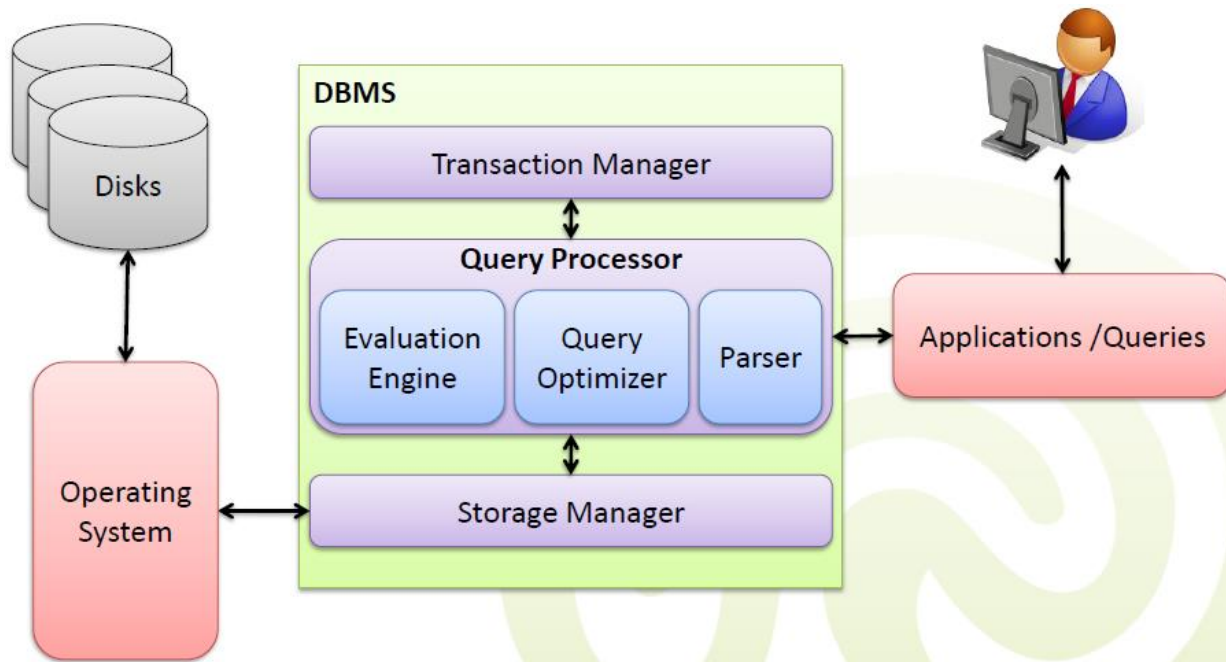


Outline

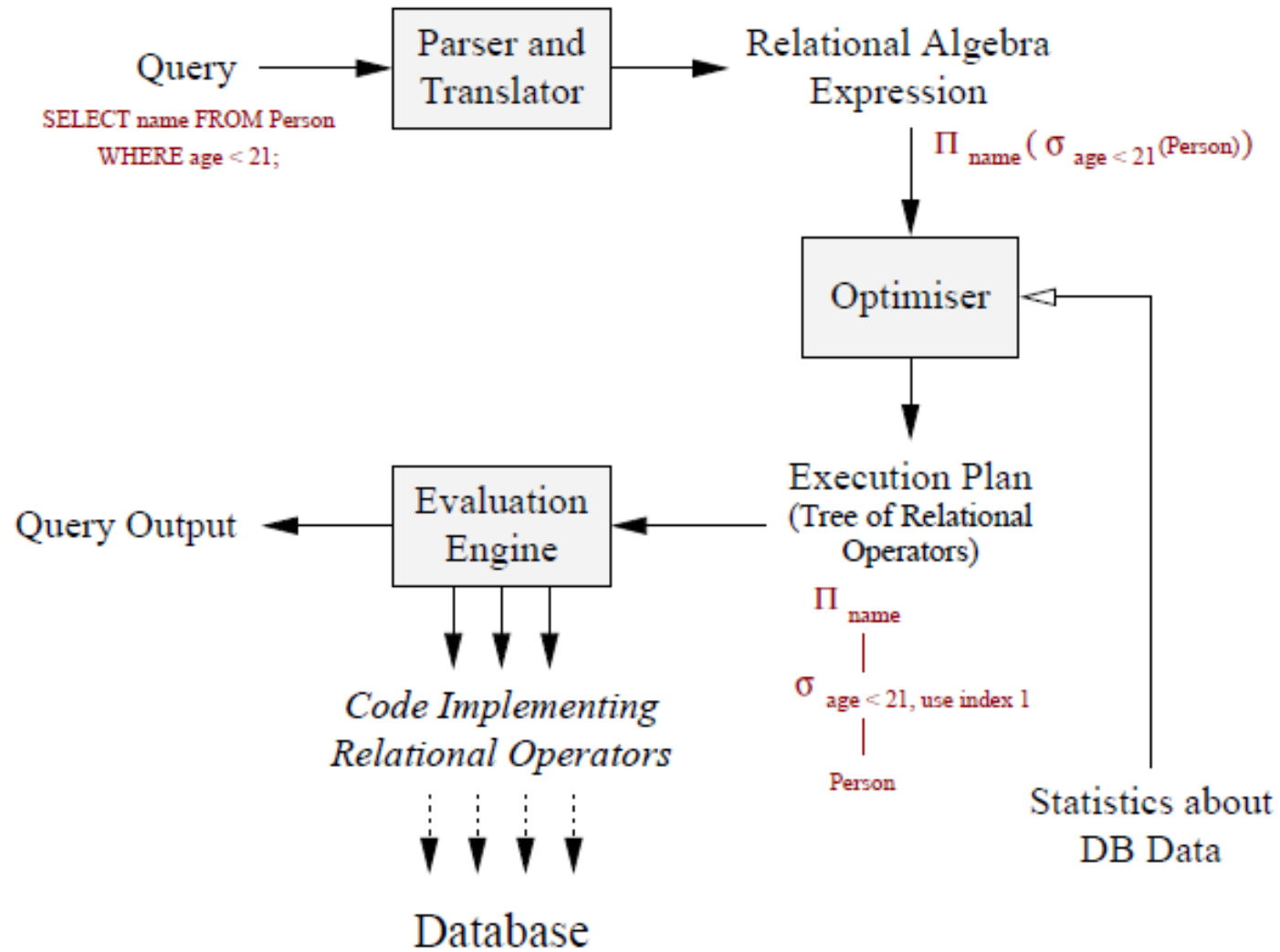
- Query Processing in DBMS
- Query optimization techniques
- Heuristic query optimization
 - Translating SQL into relational algebra
 - Reordering query operations
 - Transformation rules for relational algebra operations
- *Readings from the textbook:*
 - Chapter 19: Algorithms for Query Processing and Optimization

Query Processing in DBMS

- Users/applications submit queries to the DBMS
- The DBMS processes queries before evaluating them
 - Recall: DBMS mainly use declarative query languages (such as SQL)
 - Queries can often be evaluated in different ways
 - SQL queries do not determine how to evaluate them



Query Processing in DBMS



Query Optimisation

- Query preparation:
 - Decompose query into query blocks containing
 - Exactly one SELECT and FROM clause
 - At most one WHERE, GROUP BY and HAVING clause

SELECT *attribute*₁, ..., *attribute*_{*n*}

→ π (*attribute*₁, ..., *attribute*_{*n*})

FROM *relation*₁, ..., *relation*_{*k*}

→ (*relation*₁ × ... × *relation*_{*k*})

WHERE *condition*₁ AND/OR ... AND/OR *condition*_{*m*}

→ σ (*condition*₁ AND/OR ... AND/OR *condition*_{*m*})

- Nested queries within a query are identified as separate query blocks
- Aggregate operators in SQL must be included in the extended algebra

Heuristic Query Optimization: An Example

Student({Lname, Fname, StudId, Major}, {StudId})

Grades({StudId, CrsId, Grade}, {StudId, CrsId})

Course({Cname, CrsId, Points, Dept}, {CrsId})

- **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';
```

- **Relational Algebra:**

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

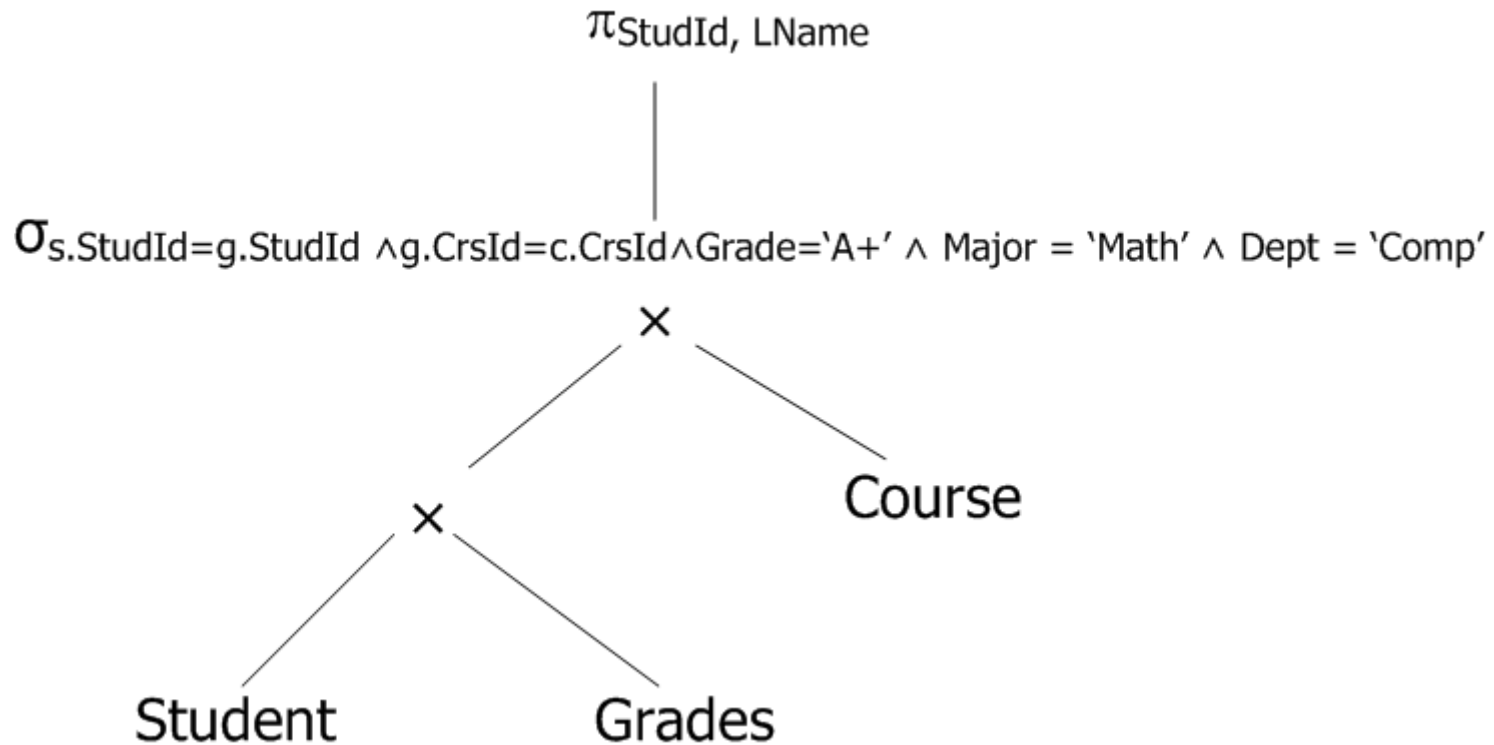
Query Tree

- **Query tree:** A tree data structure that corresponds to a relational algebra expression
 - It represents the input relations of the query as **leaf nodes** of the **tree**, and
 - Represents the relational algebra operations as internal nodes
- The initial query tree for the example SQL query:
 - Lower level nodes, starting from leaves, contain **Cartesian** product operators
 - These are applied onto relations from the SQL **FROM** clause
 - After that are (relational) **select** and **join** conditions from SQL WHERE clause to upper tree nodes applied
 - Finally is **project** operator of the SELECT clause attribute list to tree root applied

Query Tree Example

- Relational Algebra:

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



Query Tree Exercise

- Please draw query trees for the following relational algebra expressions

$\pi_{\text{StudId}} (\sigma_{\text{Grade}='A+'}(\text{Grades}))$

Query Tree Exercise

- Please draw query trees for the following relational algebra expressions

$\pi_{\text{StudId, LName}} (\sigma_{\text{Grade}='A+'} (\text{Student} * \text{Grades}))$

Query Tree Exercise

- Please draw query trees for the following relational algebra expressions

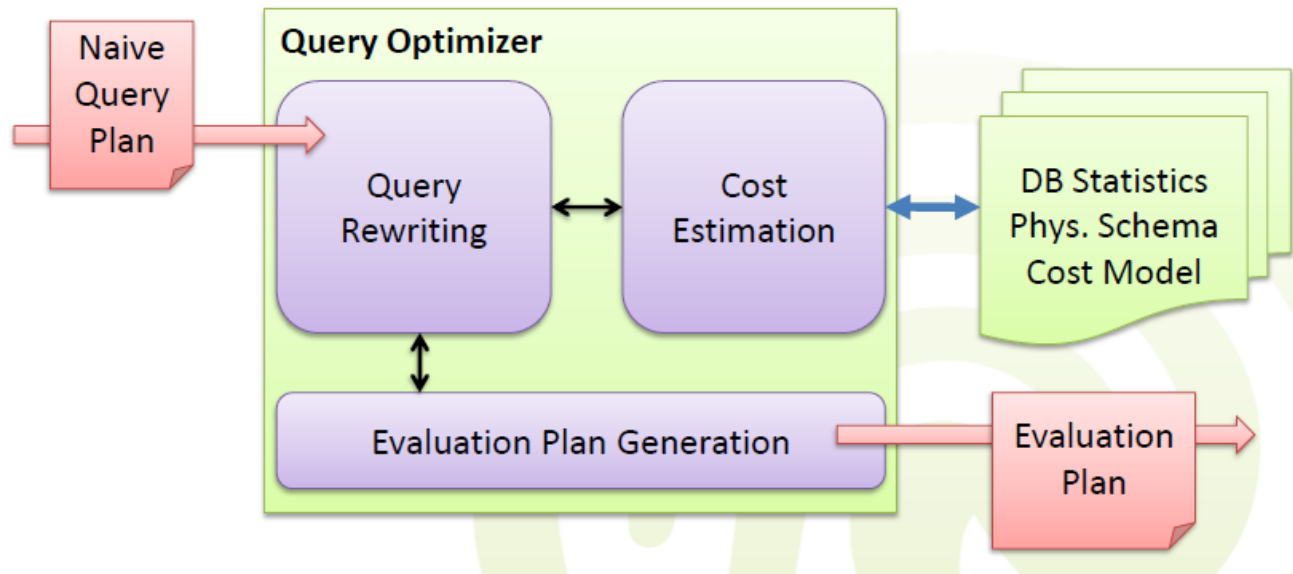
$$(\pi_{\text{StudId}}, (\sigma_{\text{CrsId} = \text{'M214'}} (\text{Grades}))) - (\pi_{\text{StudId}}, (\sigma_{\text{CrsId} = \text{'C201'}} (\text{Grades})))$$

Query Execution Plan

- For each query tree, computation proceeds bottom-up:
 - Child nodes must be executed before their parent nodes
 - But there can exist multiple methods of executing sibling nodes, e.g.
 - Process sequentially
 - Process in parallel
- A query execution plan consists of a query tree with additional annotation at each node indicating the implementation method for each RA operator
- The query optimizer determines which query execution plan is optimal, using a variety of algorithms
- Realistically, we cannot expect to always find the best plan but we expect to consistently find a plan that is good (near-optimal)

Query Optimisation

- Query optimizer rewrites the naïve (canonical) query plan into a more efficient evaluation plan



- Choosing a suitable query execution plan is called **Query Optimization** and it mainly means making decisions
 - On the order of the execution of basic relational algebra operations and
 - On data access methods

Query Optimization Techniques

- Two main query optimization techniques
 - One relies on the **heuristic** reordering of the relational algebra operations,
 - The other involves systematic estimating the **cost** of different execution plans, and choosing one with the lowest cost

- *Heuristics vs. cost-based* optimization
 - General heuristics allow to improve performance of most queries
 - Costs estimated from statistics allow for a good optimization of each specific query
 - Most DBMS use a hybrid approach between heuristics and cost estimations

Query Optimization Techniques

- Enumerating potential query plans:

Dynamically finding the best query plan is **easy**

1. Apply transformations to generate all possible plans
2. Assign total costs according to cost model
3. Choose least expensive plan



But this exhaustive search strategy for finding the best plan for each query is **prohibitively expensive**

- **Actually: not the optimal plan is needed, but the crappy plans have to be avoided**

Heuristic Query Optimization

- Process for heuristics optimization
 1. The parser of a high-level query generates an initial internal representation
 2. Apply heuristics rules to optimize the internal representation
 3. A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query

- The main heuristic is to apply first the operations that reduce the size of intermediate results
 - E.g., Apply `SELECT` and `PROJECT` operations before applying the `JOIN` or other binary operations

Using Heuristics in Query Optimization (1)

- General Transformation Rules for Relational Algebra Operations:

1. Cascade of σ : A conjunctive selection condition can be broken up into a cascade (sequence) of individual σ operations:

$$\sigma_{c_1 \wedge c_2 \wedge \dots \wedge c_n}(N) = \sigma_{c_1}(\sigma_{c_2}(\dots(\sigma_{c_n}(N))\dots))$$

$$\sigma_{\text{grade}='A+' \wedge \text{courId}='C302'}(\text{Grades}) = \sigma_{\text{grade}='A+'}(\sigma_{\text{courId}='C302'}(\text{Grades}))$$

2. Commutativity of σ : The σ operation is commutative:

$$\sigma_{c_1}(\sigma_{c_2}(N)) = \sigma_{c_2}(\sigma_{c_1}(N))$$

$$\sigma_{\text{grade}='A+'}(\sigma_{\text{courId}='C302'}(\text{Grades})) = \sigma_{\text{courId}='C302'}(\sigma_{\text{grade}='A+'}(\text{Grades}))$$

Using Heuristics in Query Optimization (1)

- General Transformation Rules for Relational Algebra Operations:
- 3. Cascade of π : In a cascade (sequence) of π operations, all but the last one can be ignored:

$$\pi_{List1} (\pi_{List2} (\dots (\pi_{Listn}(N)) \dots)) = \pi_{List1}(N)$$

$$\pi_{StudId} (\pi_{StudId, LName}(Student)) = \pi_{StudId}(Student)$$

Using Heuristics in Query Optimization (1)

- General Transformation Rules for Relational Algebra Operations:
4. Commuting σ with π : If the selection condition c involves **only** the attributes A_1, \dots, A_n in the projection list, the two operations can be commuted:
- $$\pi_{A_1, A_2, \dots, A_n} (\sigma_c (N)) = \sigma_c (\pi_{A_1, A_2, \dots, A_n} (N))$$

$$\begin{aligned} & \pi_{\text{StudId, major}} (\sigma_{\text{major}='Comp'}(\text{Student})) \\ &= \sigma_{\text{major}='Comp'}(\pi_{\text{StudId, major}}(\text{Student})) \end{aligned}$$

$$\begin{aligned} & \pi_{\text{StudId, FName}} (\sigma_{\text{major}='Comp'}(\text{Student})) \\ &= \sigma_{\text{major}='Comp'}(\pi_{\text{StudId, FName}}(\text{Student})) \end{aligned}$$

Using Heuristics in Query Optimization (2)

- General Transformation Rules for Relational Algebra Operations (contd.) (\bowtie represent both natural join $*$ and equi-join \bowtie_c):

5. Commutativity of \bowtie (and x): The \bowtie operation is commutative as is the x operation:

- $$N_1 \bowtie_c N_2 = N_2 \bowtie_c N_1; N_1 \times N_2 = N_2 \times N_1$$

$$\text{Student} \bowtie_{\text{StudId-StudId}} \text{Grades} = \text{Grades} \bowtie_{\text{StudId-StudId}} \text{Student}$$

$$\text{Student} \times \text{Grades} = \text{Grades} \times \text{Student}$$

Using Heuristics in Query Optimization (2)

6. Commuting σ with \bowtie (or \times): If all the attributes in the selection condition c involve only the attributes of one of the relations being joined—say, N_1 —the two operations can be commuted as follows:

- $$\sigma_c (N_1 \bowtie N_2) = \sigma_c (N_1) \bowtie N_2$$

$$\sigma_{\text{major}='Math'} (\text{Student} * \text{Grades}) = \sigma_{\text{major}='Math'} (\text{Student}) * \text{Grades}$$

- Alternatively, if the selection condition c can be written as (c_1 and c_2), where condition c_1 involves only the attributes of N_1 and condition c_2 involves only the attributes of N_2 , the operations commute as follows:

- $$\sigma_c (N_1 \bowtie N_2) = \sigma_{c_1} (N_1) \bowtie \sigma_{c_2} (N_2)$$

$$\begin{aligned} & \sigma_{\text{major}='Math' \wedge \text{grade}='A+'} (\text{Student} * \text{Grades}) \\ &= \sigma_{\text{major}='Math'} (\text{Student}) * \sigma_{\text{grade}='A+'} (\text{Grades}) \end{aligned}$$

Using Heuristics in Query Optimization (3)

- General Transformation Rules for Relational Algebra Operations (contd.):

7. Commuting π with \bowtie (or \times): Suppose that the projection list is $AL = \{A_1, \dots, A_n, B_1, \dots, B_m\}$, where A_1, \dots, A_n are attributes of N_1 and B_1, \dots, B_m are attributes of N_2 .

- If the join condition c involves only attributes in AL , the two operations can be commuted as follows:

$$\pi_{AL} (N_1 \bowtie_C N_2) = (\pi_{A_1, \dots, A_n} (N_1)) \bowtie_C (\pi_{B_1, \dots, B_m} (N_2))$$

Using Heuristics in Query Optimization (3)

- If the join condition C contains additional attributes not in AL , these must be added to the projection list, and a final π operation is needed.

$$\pi_{AL}(N_1 \bowtie_C N_2) = \pi_{AL}(\pi_{AL_1}(N_1) \bowtie_C \pi_{AL_2}(N_2)),$$

where AL_i contains the common attributes in N_i and AL and the common attributes of N_1 and N_2

Using Heuristics in Query Optimization (4)

- General Transformation Rules for Relational Algebra Operations (contd.):

8. Commutativity of set operations: The set operations \cup and \cap are commutative but “-” is not.

9. Associativity of \bowtie , \times , \cup , and \cap : These four operations are individually associative; that is, if θ stands for any one of these four operations (throughout the expression), we have

- $(N_1 \theta N_2) \theta N_3 = N_1 \theta (N_2 \theta N_3)$

10. Commuting σ with set operations: The σ operation commutes with \cup , \cap , and “-”. If θ stands for any one of these three operations, we have

- $\sigma_c (N_1 \theta N_2) = (\sigma_c (N_1)) \theta (\sigma_c (N_2))$

Using Heuristics in Query Optimization (5)

- General Transformation Rules for Relational Algebra Operations (contd.):

11. The π operation commutes with \cup .

$$\pi_{AL} (N_1 \cup N_2) = (\pi_{AL} (N_1)) \cup (\pi_{AL} (N_2))$$

12. Converting a (σ, x) sequence into \bowtie : If the condition c of a σ that follows a x Corresponds to a join condition, convert the (σ, x) sequence into a \bowtie as follows:

$$\sigma_c (N_1 x N_2) = r(N_1) \bowtie_c N_2$$

Using Heuristics in Query Optimization (6)

- Summary of Heuristics for Algebraic Optimization:
 1. The main heuristic is to apply first the operations that reduce the size of intermediate results.
 2. Perform select operations as early as possible to reduce the number of tuples and perform project operations as early as possible to reduce the number of attributes. (This is done by moving select and project operations as far down the tree as possible, e.g. rule 6, 7, 10, 11)
 3. The select and join operations that are most restrictive should be executed before other similar operations. (This is done by reordering the leaf nodes of the tree among themselves and adjusting the rest of the tree appropriately.)

A Sample Relational Database

Student

Lname	Fname	StudId	Major
Smith	Susan	131313	Comp
Bond	James	007007	Math
Smith	Susan	555555	Comp
Cecil	John	010101	Math

Grades

StudId	CourId	Grad
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

Course

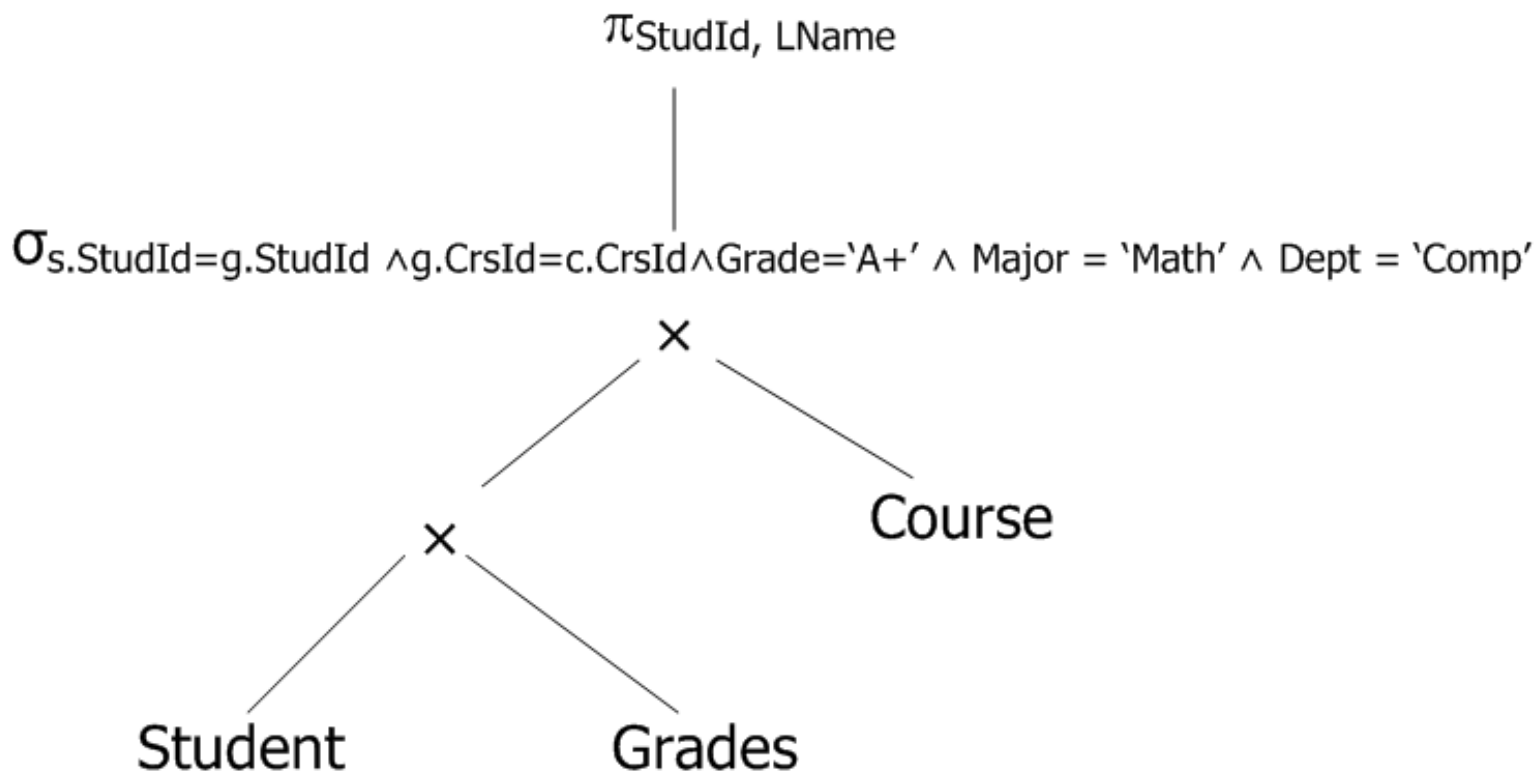
Cname	CourId	Points	Dept
DB Systems	C302	15	Comp
Software Engineering	C301	15	Comp
Discrete Math	M214	22	Math
Programmes	C201	22	Comp

An Example Initial Query Tree:

- Relational algebra query

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

- Query tree

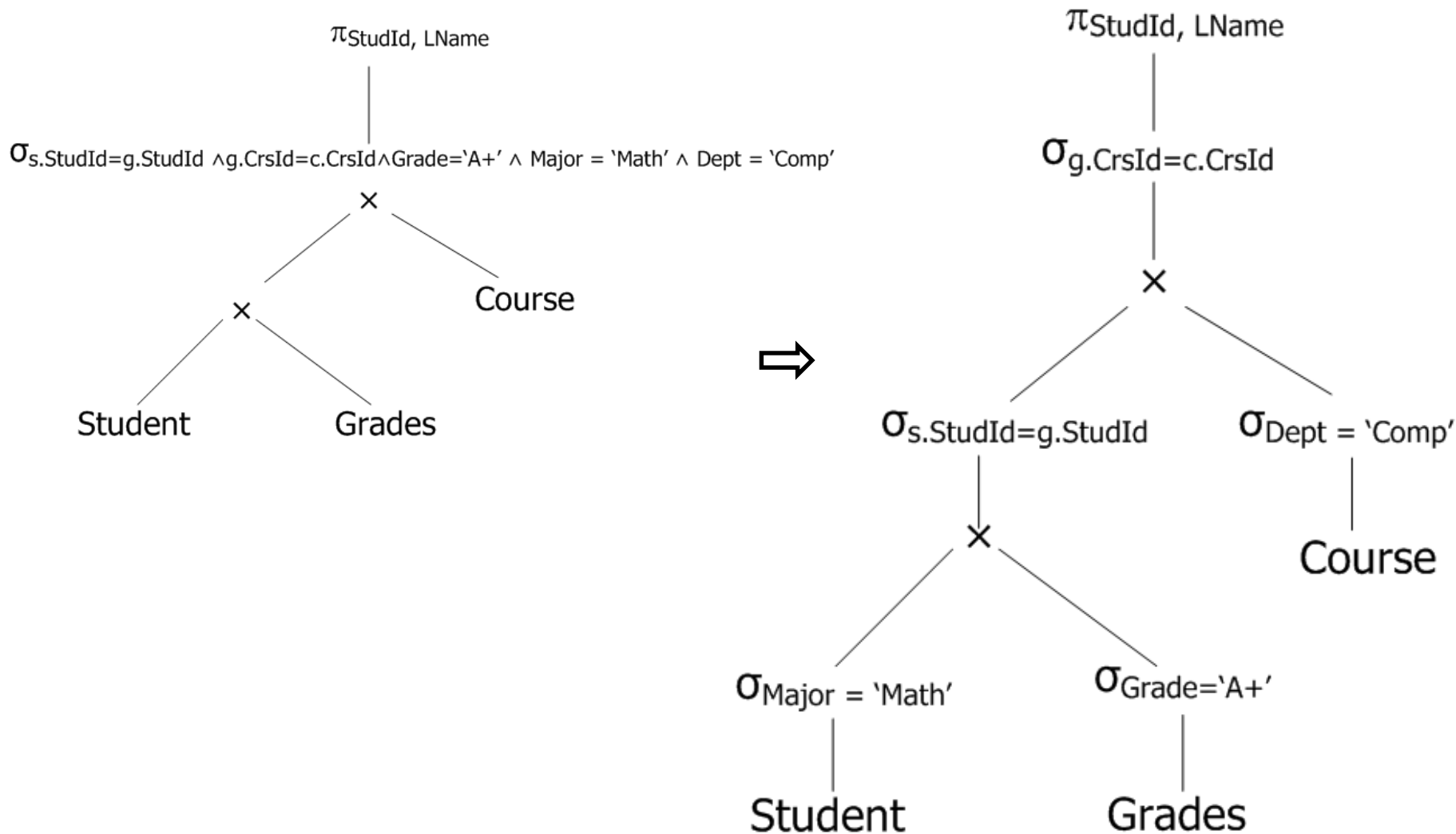


Analysis of the Initial Query Tree

- According to the structure of the initial query tree, two Cartesian products should be executed **first**
- The main heuristic: Apply SELECT and PROJECT operations before applying the JOIN or other binary operations
- Hence, **move** select operations down the tree

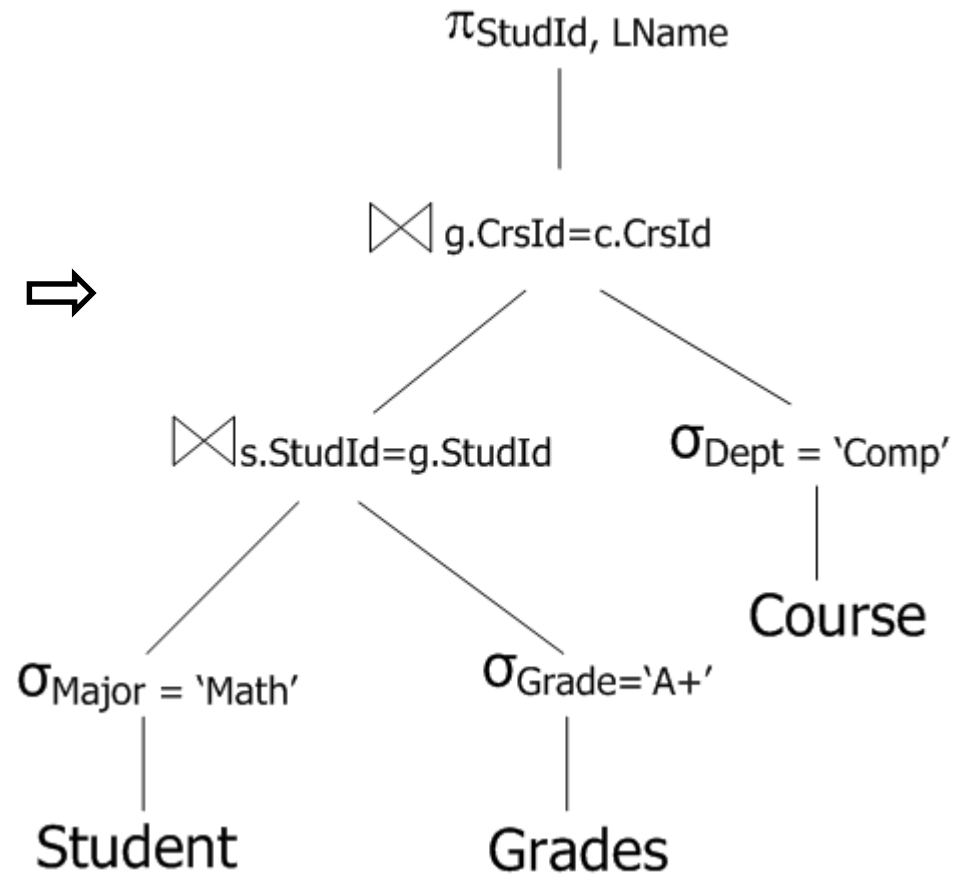
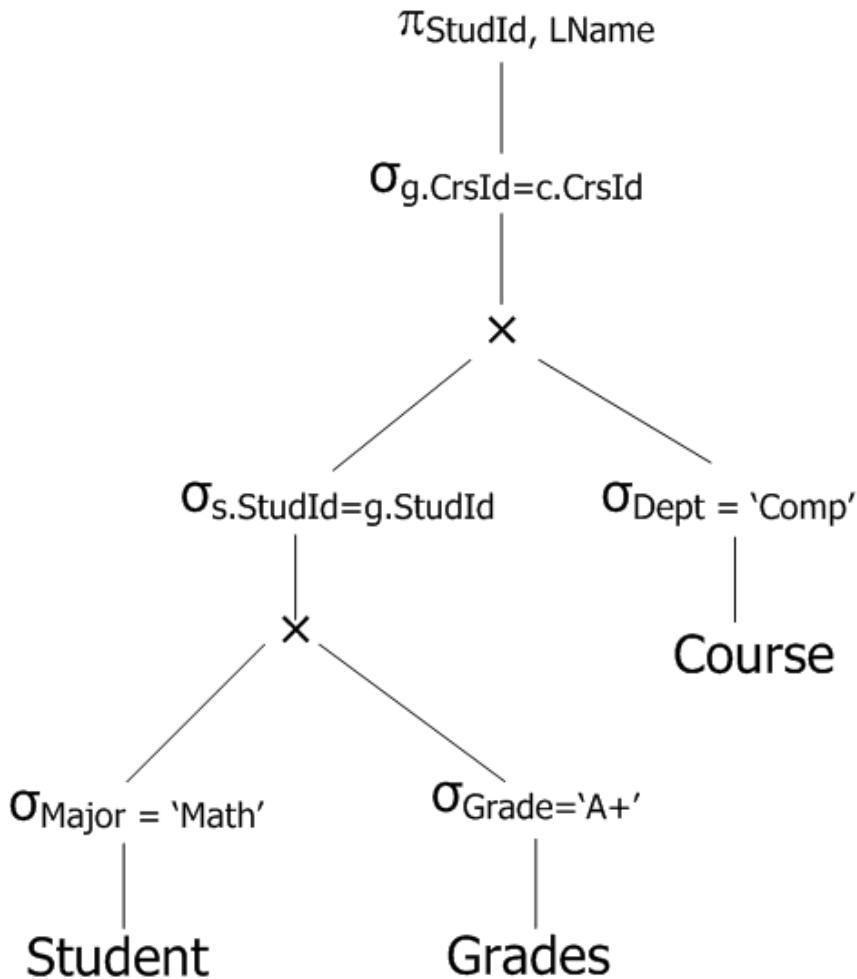
Query Tree After Moving Down Selects

- Move select operations down the tree (Rule 6)



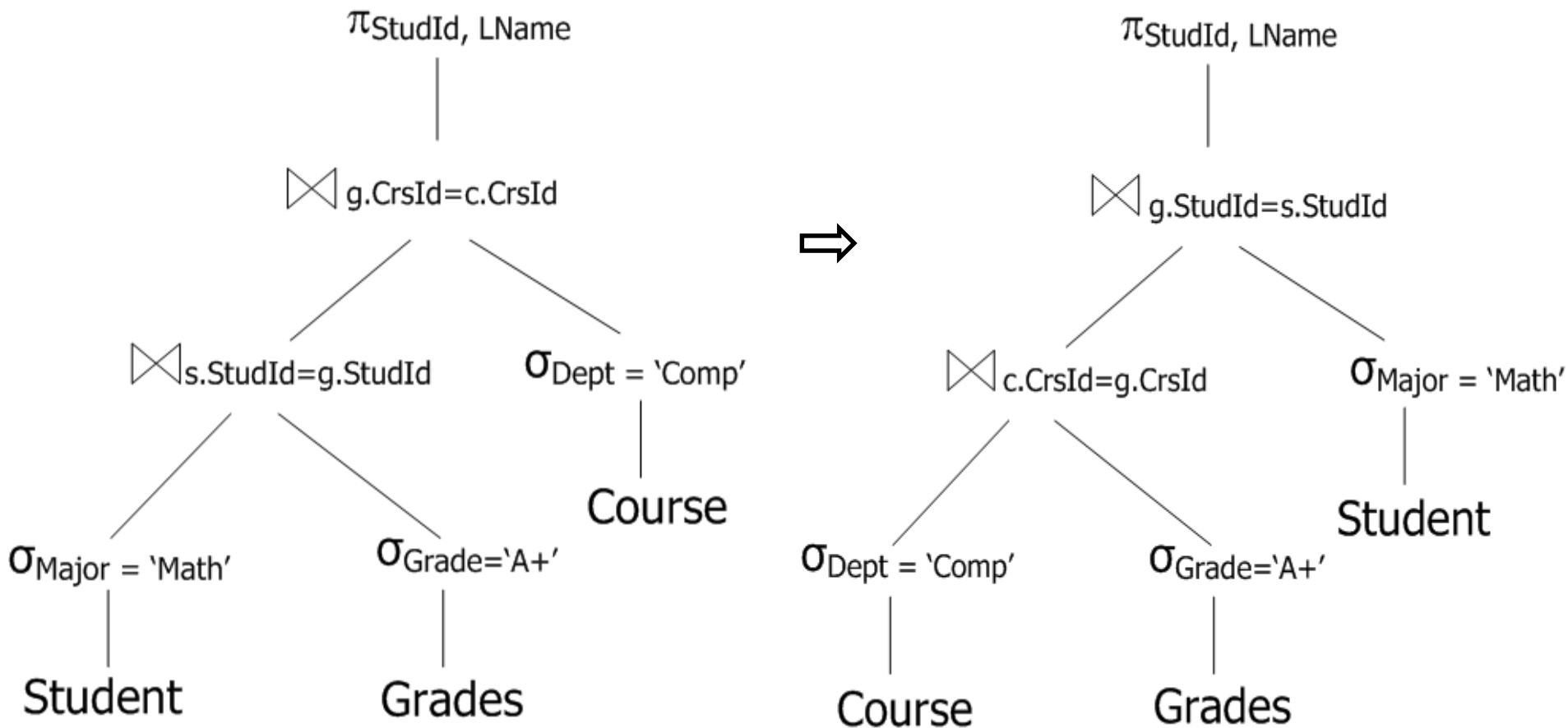
Query Tree After Introducing Joins

- Replacing each Cartesian product followed by a select according to a join condition with a **join** operator (Rule 12)



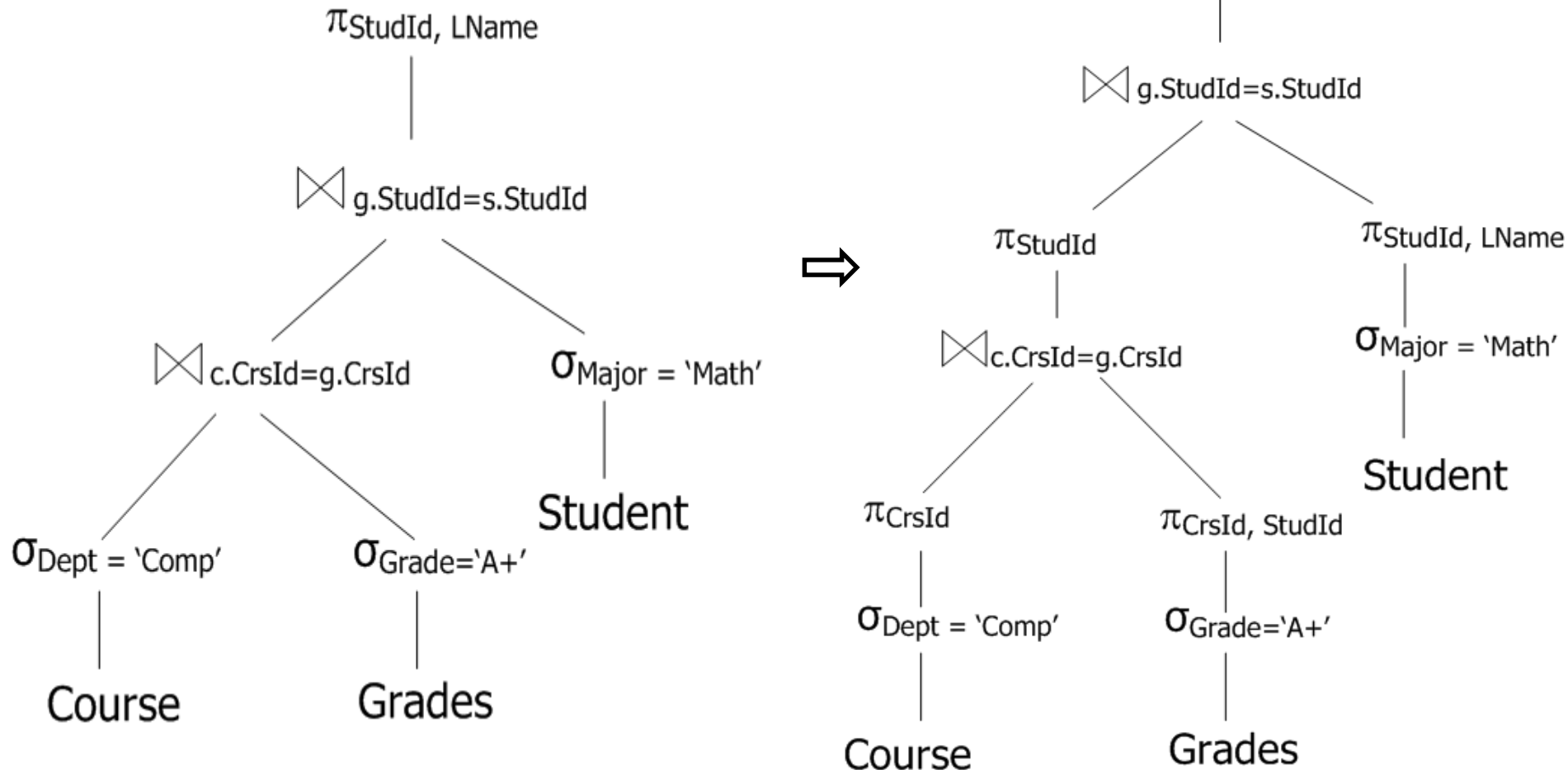
Query Tree After Switching Join Nodes

- Assume there are less Comp courses than Math major students
- switching** Course and Student so that the very **restrictive** select operation could be applied as **early** as possible (Rule 9)



Query Tree after Pushing down Project

- keeping in intermediate relations only the attributes needed by subsequent operations by applying **project** (π) operations as early as possible (Rule 7)



Effect of Project Operations

- Performing project operations as early as possible brings an improvement in the **efficiency** of a query execution
 - tuples get **shorter** after projection, more of them will fit into a file block of the same size
 - the same number of tuples will be contained in a **smaller** number of blocks
 - As there will be less blocks to be processed by subsequent operations, the query execution will be **faster**

Query Optimisation

- All transformations can be applied to the canonical evaluation plan
 - However, there is no best operator sequence that is always optimal
 - Efficiency depends on the current data instance, the actual implementation of base operations, access paths and indexes, etc.
- Idea: assign average costs to operators (nodes) and aggregate costs for each query plan

Summary

- Heuristic optimization converts a declarative query to a canonical algebraic query tree that is then gradually transformed using transformation rules
- The main heuristics is to perform unary relational operations (selection and projection) before binary operations (joins, set theoretic), and aggregate functions with (or without) grouping

Next

- Cost-based query optimization
- Readings:
 - Chapter 19: Algorithms for Query Processing and Optimization
 - Chapters 17: Disk Storage, Basic File Structures, and Hashing (Sections: 17.2, to 17.8)
 - Chapter 18: Indexing Structures for Files (Sections: 17.1 to 17.5)