Query Processing

Dr. Jyotismita Chaki

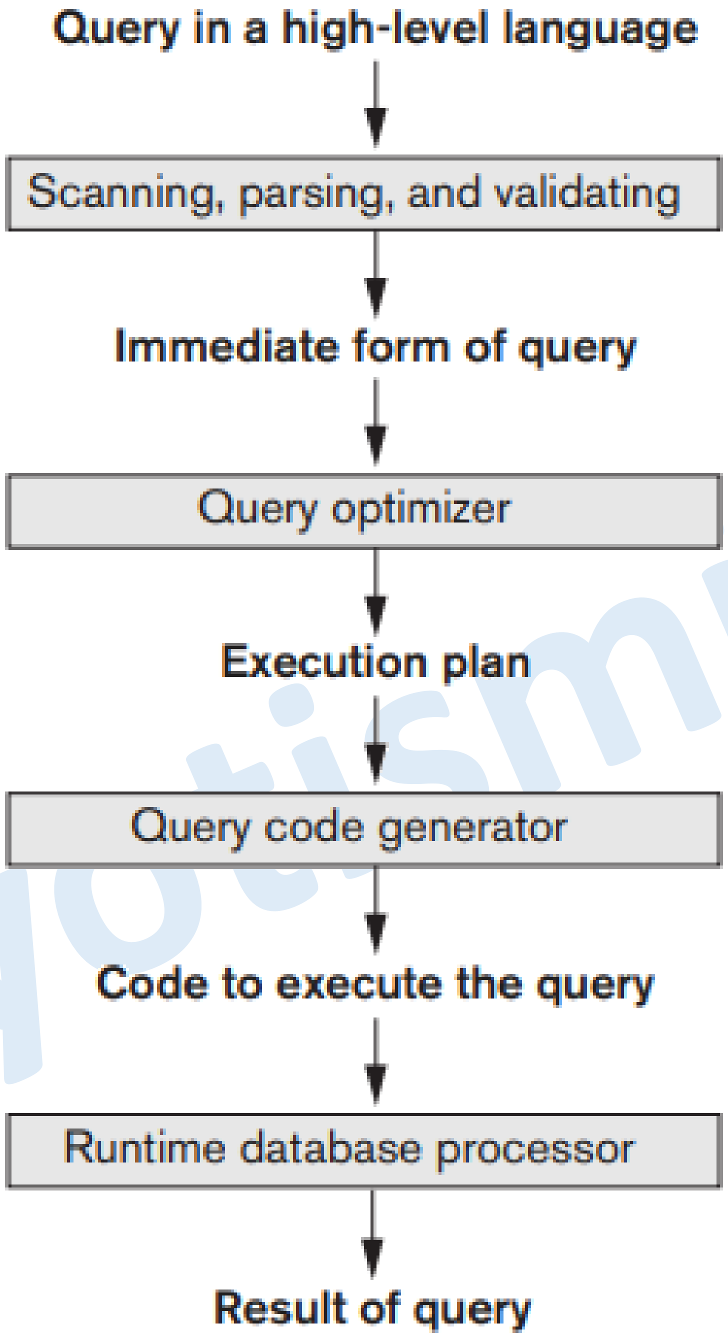
Query processing

- A query expressed in a high-level query language such as SQL must first be **scanned, parsed, and validated**.
- The **scanner** identifies the query tokens—such as SQL keywords, attribute names, and relation names—that appear in the text of the query, whereas the **parser** checks the query syntax to determine whether it is formulated according to the syntax rules (rules of grammar) of the query language.
- The query must also be validated by checking that all attribute and relation names are valid and semantically meaningful names in the schema of the particular database being queried.

Query processing

- An internal representation of the query is then created, usually as a tree data structure called a query tree.
- It is also possible to represent the query using a graph data structure called a query graph, which is generally a directed acyclic graph (DAG).
- A query has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as query optimization.

Query processing



Code can be:

Executed directly (interpreted mode)

Stored and executed later whenever needed (compiled mode)

Relational Algebra

- The basic set of operations for the formal relational model.
- These operations enable a user to specify basic retrieval requests as relational algebra expressions.
- The result of a retrieval query is a new relation.
- A sequence of relational algebra operations forms a **relational algebra expression**, whose result will also be a relation that represents the result of a database query.
- Importance:
  - Provides a formal foundation for relational model operations.
  - Used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of RDBMS.
  - Some of its concepts are incorporated into the SQL standard query language for RDBMS

Relational Algebra: Unary Relational Operations: SELECT

- The SELECT operator is unary; that is, it is applied to a single relation.
- Used to choose a subset of the tuples from a relation that satisfies a selection condition: selects some rows
- A filter that keeps only those tuples that satisfy a qualifying condition.
- Can also be visualized as a horizontal partition of the relation into two sets of tuples
- Denoted by: $\boldsymbol{\sigma}_{<\textbf{select condition}>}\mathbf{(R)}$ [σ (sigma): SELECT operator, selection condition: Boolean expression (condition) specified on the attributes of relation R, R: relational algebra expression whose result is a relation]
- For example, to select the EMPLOYEE tuples whose department is 4: $\boldsymbol{\sigma}_{\textbf{Dno = 4}}\textbf{(EMPLOYEE)}$
- $|\sigma_c (R)| \leq |R|$ for any condition C

Relational Algebra: Unary Relational Operations: SELECT

- The Boolean expression specified in is made up of a number of clauses of the form:
    - <attribute name> < comparison op> <constant value>
    - <attribute name> < comparison op> < attribute name>
    - where is the name of an attribute of R, is normally one of the operators {=, <, ≤, >, ≥, ≠}, and is a constant value from the attribute domain.
- For example, to select the tuples for all employees who either work in department 4 and make over \$25,000 per year, or work in department 5 and make over \$30,000, we can specify the following SELECT operation: $\boldsymbol{\sigma}_{\textbf{(Dno=4 AND Salary>25000) OR (Dno=5 AND Salary>30000)}}\textbf{(EMPLOYEE)}$

Relational Algebra: Unary Relational Operations: SELECT

- {=, <, ≤, >, ≥, ≠} can apply to attributes whose domains are ordered values: numeric or date domains
- If the domain of an attribute is a set of unordered values, then only the comparison operators in the set {=, ≠} can be used.
- An example of an unordered domain is the domain Color = { 'red', 'blue', 'green', 'white', 'yellow', …}, where no order is specified among the various colors.
- The Boolean conditions AND, OR, and NOT have their normal interpretation, as follows:
  - (cond1 AND cond2) is TRUE if both (cond1) and (cond2) are TRUE; otherwise, it is FALSE.
  - (cond1 OR cond2) is TRUE if either (cond1) or (cond2) or both are TRUE; otherwise, it is FALSE.
  - (NOT cond) is TRUE if cond is FALSE; otherwise, it is FALSE.

Relational Algebra: Unary Relational Operations: SELECT

- The fraction of tuples selected by a selection condition is referred to as the **selectivity** of the condition.
- SELECT operation is commutative
  - $\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$
- We can always combine a cascade (or sequence) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition
  - $\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots(\sigma_{<condn>}(R))\ldots)) = \sigma_{<cond1>}$ AND $\sigma_{<cond2>}$ AND...AND $\sigma_{<condn>}(R)$
- $\sigma_{(Dno=4 \text{ AND } Salary>25000) \text{ OR } (Dno=5 \text{ AND } Salary>30000)}$ (EMPLOYEE)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

Relational Algebra: Unary Relational Operations: PROJECT

- Selects certain columns (or attributes) from the table and discards the other columns.
- The result can be visualized as a vertical partition of the relation into two relations:
  - the needed columns (attributes)
  - the discarded columns
- The general form of the PROJECT operation: $\boldsymbol{\pi}_{<\text{attribute\_list}>}\mathbf{(R)}$**,** where π (pi) is the symbol used to represent the PROJECT operation, and is the desired sublist of attributes from the attributes of relation R.
- Its degree is equal to the number of attributes in <attribute list>
- For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows: $\pi_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$
- **Commutativity** does not hold on PROJECT.

Relational Algebra: Unary Relational Operations: PROJECT

- PROJECT operation can do **duplicate elimination.**
- For example, consider the following PROJECT operation: $\pi_{Sex, Salary}$(EMPLOYEE)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Relational Algebra: In-line expression

- To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a SELECT and a PROJECT operation.
  - $\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}}=5(\text{EMPLOYEE}))$

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Relational Algebra: Unary Relational Operations: Rename

- We can explicitly show the sequence of operations, giving a name to each intermediate relation, and using the assignment operation, denoted by ← (left arrow), as follows:
  - DEP5_EMPS ← $\sigma_{Dno=5}$(EMPLOYEE)
  - RESULT ← $\pi_{Fname,\ Lname,\ Salary}$(DEP5_EMPS)
- To rename the attributes in a relation, we simply list the new attribute names in parentheses, as in the following example:
  - TEMP ← $\sigma_{Dno=5}$(EMPLOYEE)
  - R(First_name, Last_name, Salary) ← $\pi_{Fname,\ Lname,\ Salary}$(TEMP)

TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Relational Algebra: Unary Relational Operations: Rename

- We can rename either the relation name or the attribute names, or both—as a unary operator.
- The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms:
    - $\rho_{S(B1,\ B2,\ \ldots,\ Bn)}(R)$: renames both the relation and its attributes
    - $\rho_S(R)$: renames the relation only
    - $\rho_{(B1,\ B2,\ \ldots,\ Bn)}(R)$: renames the attributes only
    - where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name, and B1, B2, … , Bn are the new attribute names.

Relational Algebra: UNION

- The result of this operation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S.
- Duplicate tuples are eliminated.
- For example, to retrieve the Social Security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5:
  - DEP5_EMPS ← σ_{Dno}=5(EMPLOYEE)
  - RESULT1 ← π_{Ssn}(DEP5_EMPS)
  - RESULT2(Ssn) ← π_{Super_ssn}(DEP5_EMPS)
  - RESULT ← RESULT1 ∪ RESULT2

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Relational Algebra: INTERSECTION and MINUS

- INTERSECTION: The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S.
- SET DIFFERENCE (or MINUS): The result of this operation, denoted by $R – S$, is a relation that includes all tuples that are in R but not in S.
- UNION and INTERSECTION are commutative operations:
  - $R \cup S = S \cup R$ and
  - $R \cap S = S \cap R$
- UNION and INTERSECTION are associative operations:
  - $R \cup (S \cup T) = (R \cup S) \cup T$ and
  - $(R \cap S) \cap T = R \cap (S \cap T)$
- MINUS operation is not commutative:
  - $R - S \neq S - R$

Relational Algebra: Example: Union, Intersection, Minus

(a) **STUDENT**

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

(a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

Relational Algebra: Cartesian Product / Cartesian Join

- Denoted by ×
- Produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set).
- Result of R(A1, A2, … , An) × S(B1, B2, … , Bm) is Q(A1, A2, … , An, B1, B2, … , Bm): degree n + m attributes
- If $|R| = n_R$ and $|R| = n_S$, then $|R \times S| = n_R * n_S$

Relational Algebra: Cartesian Product / Cartesian Join: Example

- We want to retrieve a list of names of each female employee's dependents
  - FEMALE_EMPS $\leftarrow \sigma_{Sex='F'}$(EMPLOYEE)
  - EMPNAMES $\leftarrow \pi_{Fname,\ Lname,\ Ssn}$(FEMALE_EMPS)
  - EMP_DEPENDENTS $\leftarrow$ EMPNAMES $\times$ DEPENDENT
  - ACTUAL_DEPENDENTS $\leftarrow \sigma_{Ssn=Essn}$(EMP_DEPENDENTS)
  - RESULT $\leftarrow \pi_{Fname,\ Lname,\ Dependent\_name}$(ACTUAL_DEPENDENTS)

Relational Algebra: Binary relational operations: JOIN / INNER JOIN

- Denoted by $\bowtie$
- Used to combine related tuples from two relations into single "longer" tuples.
- Suppose that we want to retrieve the name of the manager of each department.
- The general form of a JOIN operation on two relations5 R(A1, A2, … , An) and S(B1, B2, … , Bm) is $R \bowtie_{<\text{join condition}>} S$
- The result of the JOIN is a relation Q with n + m attributes Q(A1, A2, … , An, B1, B2, … , Bm) in that order
- To get the manager's name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr_ssn value in the department tuple.
- We do this by using the JOIN operation and then projecting the result over the necessary attributes, as follows:
  - $\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE}$
  - $\text{RESULT} \leftarrow \pi_{\text{Dname, Lname, Fname}}(\text{DEPT\_MGR})$

Relational Algebra: Binary relational operations: OUTER JOIN

- A set of operations, called **outer joins**, were developed for the case where the user wants to keep all the tuples in R, or all those in S, or all those in both relations in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation.
- This satisfies the need of queries in which tuples from two tables are to be combined by matching corresponding rows, but without losing any tuples for lack of matching values.
- For example, suppose that we want a list of all employee names as well as the name of the departments they manage *if they happen to manage a department*; if they do not manage one, we can indicate it with a NULL value.

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

Relational Algebra: Variations of OUTER JOIN

- Left outer join:
  - The LEFT OUTER JOIN operation keeps every tuple in the first, or left, relation R in R ⟕ S; if no matching tuple is found in S, then the attributes of S in the join result are filled or padded with NULL values.

$$\text{TEMP} \leftarrow (\text{EMPLOYEE} ⟕_{\text{Ssn=Mgr\_ssn}} \text{DEPARTMENT})$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}(\text{TEMP})$$

- Right outer join:
  - Keeps every tuple in the second, or right, relation S in the result of R ⟖ S.
- Full outer join:
  - Keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with NULL values as needed. denoted by ⟗.

Translating queries into relational algebra

- In practice, SQL is the query language that is used in most commercial RDBMSs.
- An SQL query is first translated into an equivalent extended relational algebra expression—represented as a query tree data structure—that is then optimized.
- Typically, SQL queries are decomposed into query blocks, which form the basic units that can be translated into the algebraic operators and optimized.
- A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clauses if these are part of the block.
- Hence, nested queries within a query are identified as separate query blocks.
- Because SQL includes aggregate operators—such as MAX, MIN, SUM, and COUNT—these operators must also be included

Translating queries into relational algebra

- Throughout these notes we will use the following example database schema about movies.
- The attributes of the primary key are underlined.
  - Movie(title: string, year: int, length: int, genre: string, studioName: string, producerC#: int)
  - MovieStar(name: string, address: string, gender: char, birthdate: date)
  - StarsIn(movieTitle: string, movieYear: string, starName: string)
  - MovieExec(name: string, address: string, CERT#: int, netWorth: int)
  - Studio(name: string, address: string, presC#: int)

Translating queries into relational algebra

- Consider a general SELECT-FROM-WHERE statement of the form:
  - SELECT Select-list FROM R1, . . . , R2 T2, . . .
    WHERE Where-condition
- When the statement does not use subqueries in its where-condition, we can easily translate it into the relational algebra as follows:
  - $\pi_{\text{Select-list}}\ \sigma_{\text{Where-condition}}(R1 \times \cdots \times \rho_{T2}\ (R2) \times \cdots)$.
  - An alias R2 T2 in the FROM-clause corresponds to a renaming $\rho_{T2}\ (R2)$.
  - It is possible that there is no WHERE clause. In that case, it is of course unnecessary to include the selection σ in the relational algebra expression.
- If we omit the projection (π) we obtain the translation of the following special case:
  - SELECT * FROM R1, . . . , R2 T2, . . . WHERE Where-condition

Translating queries into relational algebra

- Consider the following SELECT-FROM-WHERE statement.
  - SELECT movieTitle FROM StarsIn, MovieStar
    WHERE starName = name AND birthdate = 1960
- Its translation is as follows:
  - $\pi_{movieTitle}\ \sigma_{starName=name\ \wedge birthdate=1960}\ (StarsIn \times MovieStar)$

Translating queries into relational algebra: Normalization

- Subqueries can occur in the WHERE clause through the operators =, , <=, >=, <>; through the quantifiers ANY, or ALL; or through the operators EXISTS and IN and their negations NOT EXISTS and NOT IN. We can easily rewrite all of these cases using only EXISTS and NOT EXISTS.
- The SQL-statement:
  - SELECT movieTitle FROM StarsIn WHERE starName IN (SELECT name FROM MovieStar WHERE birthdate = 1960)
- Can be rewritten equivalently as
  - SELECT movieTitle FROM StarsIn WHERE EXISTS (SELECT name FROM MovieStar WHERE birthdate = 1960 AND name = starName)

Translating queries into relational algebra: Normalization

- The SQL-statement
  - SELECT name FROM MovieExec WHERE netWorth >= ALL (SELECT E.netWorth FROM MovieExec E)
- Can be rewritten equivalently as
  - SELECT name FROM MovieExec WHERE NOT EXISTS (SELECT E.netWorth FROM MovieExec E WHERE netWorth < E.netWorth)
- Consider relations R(A, B) and S(C). Then
  - SELECT C FROM S WHERE C IN (SELECT SUM(B) FROM R GROUP BY A)
- Can be rewritten as
  - SELECT C FROM S WHERE EXISTS (SELECT SUM(B) FROM R GROUP BY A HAVING SUM(B) = C)

Translating queries into relational algebra: Context Relation

- To translate a query with subqueries into the relational algebra, it seems a logical strategy to work by recursion: first translate the subqueries and then combine the translated results into a translation for the entire SQL statement.
- If the subqueries contain subqueries themselves, we again translate the latter first — continuing recursively until we reach a level that does not contain subqueries.
- The subquery can refer to attributes of relations appearing in the FROM list of one of the outer lying queries. This is known as **correlated subqueries**.

Translating queries into relational algebra:

- The following query contains a subquery that refers to the starName attribute of the outer relation StarsIn.
  - SELECT movieTitle FROM StarsIn WHERE EXISTS (SELECT name FROM MovieStar WHERE birthdate = 1960 AND name = starName)
- We call the outer relations from which a correlated subquery uses certain attributes **context relations** for the subquery.
- We call the attributes of the context relations the **parameters** of the subquery.
- In the above example, *StarsIn* is hence a context relation for the subquery.
- The corresponding parameters are all attributes of StarsIn, i.e., *movieTitle, movieYear*, and *starName*.

Translating queries into relational algebra:

- To translate a SELECT– FROM–WHERE statement that is used as a subquery we must make the following rules:
  - We must add all context relations to the cartesian product of the relations in the FROM list;
  - We must add all parameters as attributes to the projection π
- The subquery:
  - SELECT movieTitle FROM StarsIn WHERE EXISTS (SELECT name FROM MovieStar WHERE birthdate = 1960 AND name = starName)
- Can be written as:
  - $\pi_{movieTitle,movieYear,starName,name}\ \sigma_{birthdate=1960\ \wedge name=starName}$ (StarsIn × MovieStar).

Translating queries into relational algebra: Cross join

- Consider the relations R(A, B) and S(C), as well as the following query:
  - SELECT S1.C, S2.C FROM S S1, S S2
    WHERE EXISTS (
    (SELECT R1.A, R1.B FROM R R1
    WHERE A = S1.C AND B = S2.C)
    CROSS JOIN
    (SELECT R2.A, R2.B FROM R R2
    WHERE B = S1.C))
- Let us translate its EXISTS subquery containing the cross join. The translations of Q1 and Q2 are as follows.
  - $E1 = \pi_{S1.C,S2.C,R1.A,R1.B}\ \sigma_{A=S1.C \wedge B=S2.C}\ (\rho_{R1}\ (R) \times \rho_{S1}\ (S) \times \rho_{S2}\ (S))$
  - $E2 = \pi_{S1.C,R2.A,R2.B}\ \sigma_{B=S1.C}\ (\rho_{R2}\ (R) \times \rho_{S1}\ (S))$
- Notice that Q1 and Q2 have one context relation in common, namely S1.
- The translation of the EXISTS subquery is then: E1 $\bowtie$ E2.

Translating queries into relational algebra: Theta join

- The join expression is not a correlated subquery, and hence we do not need to take into account context relations.
  - Movie *JOIN* StarsIn ON title = movieTitle AND year = movieYear
- It is translated as follows:
  - Movie $\bowtie_{title=movieTitle \wedge year=movieYear}$ StarsIn

Translating queries into relational algebra: Union

- Consider the relations R(A, B) and S(C), as well as the following query:
  - SELECT S1.C, S2.C FROM S S1, S S2
    WHERE EXISTS (
    (SELECT R1.A, R1.B FROM R R1
    WHERE A = S1.C AND B = S2.C)
    UNION
    (SELECT R1.A, R1.B FROM R R1
    WHERE B = S1.C))
- Let us translate its EXISTS subquery containing the cross join. The translations of Q1 and Q2 are as follows.
  - $E1 = \pi_{S1.C,S2.C,R1.A,R1.B}\ \sigma_{A=S1.C\ \wedge B=S2.C}\ (\rho_{R1}\ (R) \times \rho_{S1}\ (S) \times \rho_{S2}\ (S))$
  - $E2 = \pi_{S1.C,R1.A,R1.B}\ \sigma_{B=S1.C}\ (\rho_{R1}\ (R) \times \rho_{S1}\ (S))$
  - $\pi_{S1.C,S2.C,R1.A \rightarrow A,R1.B \rightarrow B}(E1) \cup \pi_{S1.C,S2.C,R1.A \rightarrow A,R1.B \rightarrow B}(E2 \times \rho_{S2}\ (S))$

  [V1, . . . , Vm be the context relations of Q1 that do not occur in Q2; W1, . . . , Wn be the context relations of Q2 that do not occur in Q1. We then translate SQL-expression as: π...(E1 × W1 × · · · × Wn) ∪ π...(E2 × V1 × · · · × Vm)]

Translating queries into relational algebra: Except

- Consider the relations R(A, B) and S(C), as well as the following query:
  - SELECT S1.C, S2.C FROM S S1, S S2
    WHERE EXISTS (
    (SELECT R1.A, R1.B FROM R R1
    WHERE A = S1.C AND B = S2.C)
    EXCEPT
    (SELECT R1.A, R1.B FROM R R1
    WHERE B = S1.C))
- Let us translate its EXISTS subquery containing the cross join. The translations of Q1 and Q2 are as follows.
  - $E1 = \pi_{S1.C,S2.C,R1.A,R1.B}\ \sigma_{A=S1.C \wedge B=S2.C}\ (\rho_{R1}\ (R) \times \rho_{S1}\ (S) \times \rho_{S2}\ (S))$
  - $E2 = \pi_{S1.C,R1.A,R1.B}\ \sigma_{B=S1.C}\ (\rho_{R1}\ (R) \times \rho_{S1}\ (S))$
  - $\pi_{S1.C,S2.C,R1.A \rightarrow A,R1.B \rightarrow B}(E1) - \pi_{S1.C,S2.C,R1.A \rightarrow A,R1.B \rightarrow B}(E2 \times \rho_{S2}\ (S))$

  [V1, . . . , Vm be the context relations of Q1 that do not occur in Q2; W1, . . . , Wn be the context relations of Q2 that do not occur in Q1. We then translate SQL-expression as: π...(E1 × W1 × · · · × Wn) - π...(E2 × V1 × · · · × Vm)]

Translating queries into relational algebra: Except

- Here is a simple example of an EXCEPT expression.
- This is not a correlated subquery, and hence there is no need to take context relations into account.
  - (SELECT name, address from MovieStar)
    EXCEPT
    (SELECT name, address from MovieExec)
- Its translation is
  - $(\pi_{name,address}(MovieStar) - \pi_{name,address}(MovieExec))$.

Translating queries into relational algebra: Group by and Having

- Let A1, . . . , An be the parameters of the statement (if it occurs as a subquery).
- The translation of the entire statement then is the following.
  - $\pi_{A1,...,An,Select\text{-}list}\ \sigma_{Having\text{-}condition}\ \gamma_{A1,...,An,Group\text{-}list,Agg\text{-}list}(E)$.
- Here, 'Agg-list' consists of all aggregation operations performed in the Having condition or Select-list.
- If the HAVING clause is absent, then the σ can be omitted.

Translating queries into relational algebra: Group by and Having

- The SQL statement
  - SELECT name, SUM(length)
    FROM MovieExec, Movie
    WHERE cert# = producerC#
    GROUP BY name
    HAVING MIN(year) < 1930
- Translated into
  - $\pi_{name,SUM(length)}\ \sigma_{MIN(year)<1930}\ \gamma_{name,MIN(year),SUM(length)}\ \sigma_{cert\#=producerC\#}(MovieExec \times Movie)$.

Query Tree

- A query tree is a tree data structure that corresponds to an extended relational algebra expression.
- It represents the input relations of the query as leaf nodes of the tree, and it represents the relational algebra operations as internal nodes.
- An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.
- The order of execution of operations starts at the leaf nodes, which represents the input database relations for the query, and ends at the root node, which represents the final operation of the query.
- The execution terminates when the root node operation is executed and produces the result relation for the query.

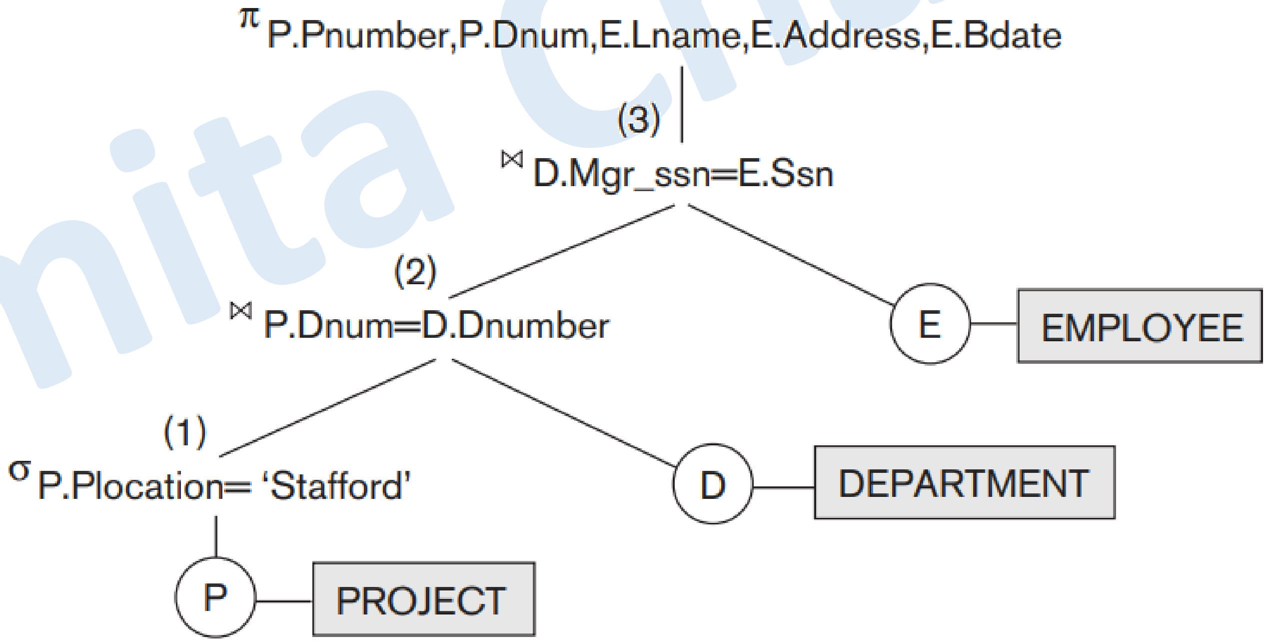
Query Tree

- **SELECT** P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate **FROM** PROJECT P, DEPARTMENT D, EMPLOYEE E **WHERE** P.Dnum=D.Dnumber AND D.Mgr_ssn=E.Ssn AND P.Plocation= 'Stafford'.
- Translation:

$$\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}\ (((\sigma_{\text{Plocation='Stafford'}}(\text{PROJECT}))$$
$$\bowtie_{\text{Dnum=Dnumber}}(\text{DEPARTMENT})) \bowtie_{\text{Mgr\_ssn=Ssn}}(\text{EMPLOYEE}))$$

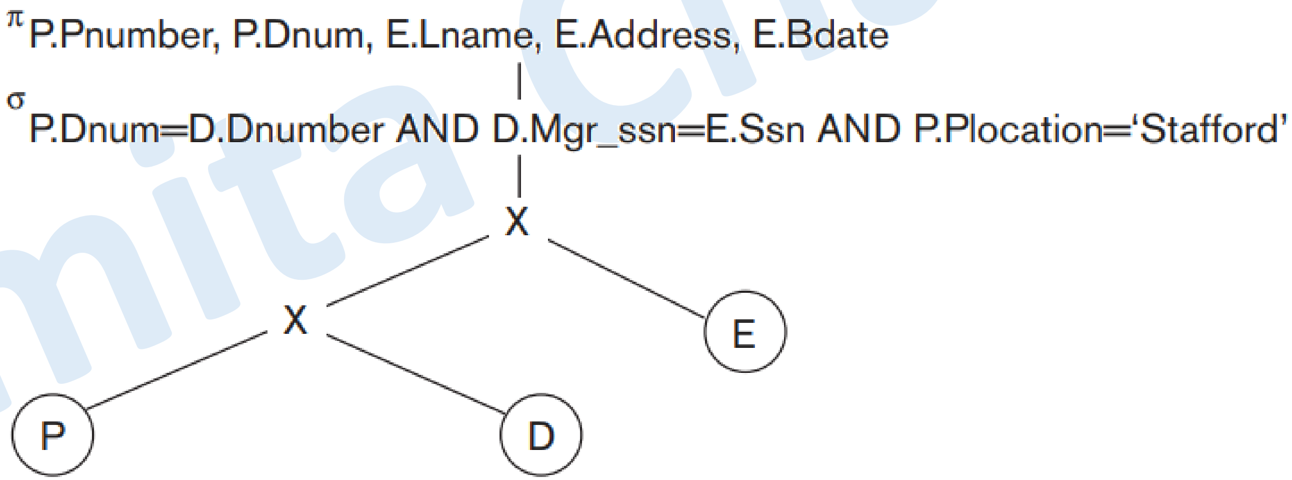
Query Tree

- In the Figure, the leaf nodes P, D, and E represent the three relations PROJECT, DEPARTMENT, and EMPLOYEE, respectively, and the internal tree nodes represent the relational algebra operations of the expression. When this query tree is executed, the node marked (1) in the Figure must begin execution before node (2) because some resulting tuples of operation (1) must be available before we can begin executing operation (2).
- Similarly, node (2) must begin executing and producing results before node (3) can start execution, and so on.



Heuristic Query optimization

- In general, many different relational algebra expressions—and hence many different query trees—can be semantically equivalent; that is, they can represent the same query and produce the same results.
- The query parser will typically generate a standard **initial query tree** to correspond to an SQL query, without doing any optimization.



- The CARTESIAN PRODUCT of the relations specified in the FROM clause is first applied; then the selection and join conditions of the WHERE clause are applied, followed by the projection on the SELECT clause attributes.

Heuristic Query optimization

- Such a **canonical query tree** represents a relational algebra expression that is very inefficient if executed directly, because of the CARTESIAN PRODUCT (×) operations.
- For example, if the PROJECT, DEPARTMENT, and EMPLOYEE relations had record sizes of 100, 50, and 150 bytes and contained 100, 20, and 5,000 tuples, respectively, the result of the CARTESIAN PRODUCT would contain 10 million tuples of record size 300 bytes each.
- However, this canonical query tree in Figure is in a simple standard form that can be easily created from the SQL query. It will never be executed.
- The **heuristic query optimizer** will transform this initial query tree into an equivalent final query tree that is efficient to execute.

Heuristic Query optimization

Student({Lname, Fname, <u>StudId</u>, Major})

Grades({<u>StudId</u>, <u>CrsId</u>, Grade})

Course({Cname, <u>CrsId</u>, Points, Dept})

- 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';
```

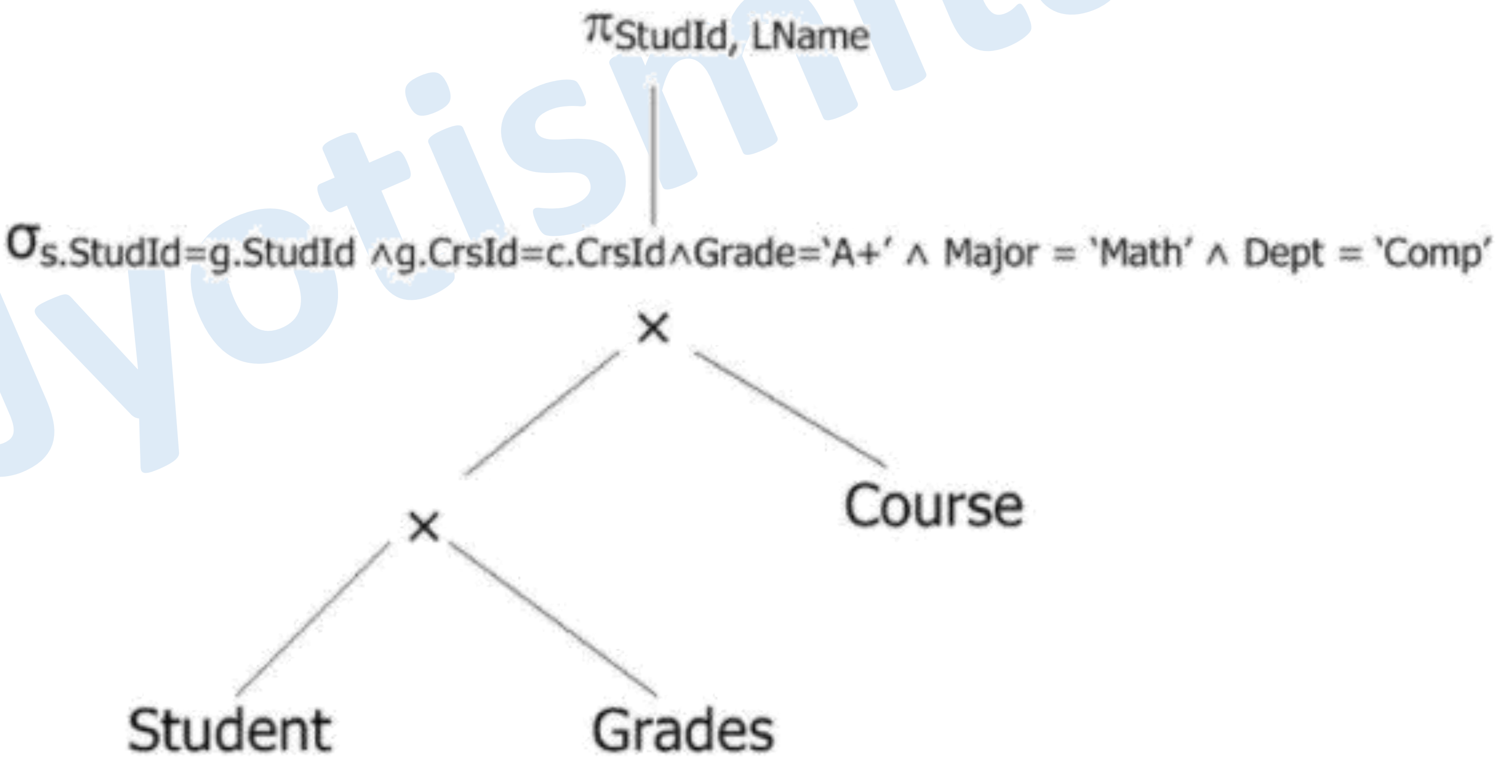
- Relational Algebra:

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

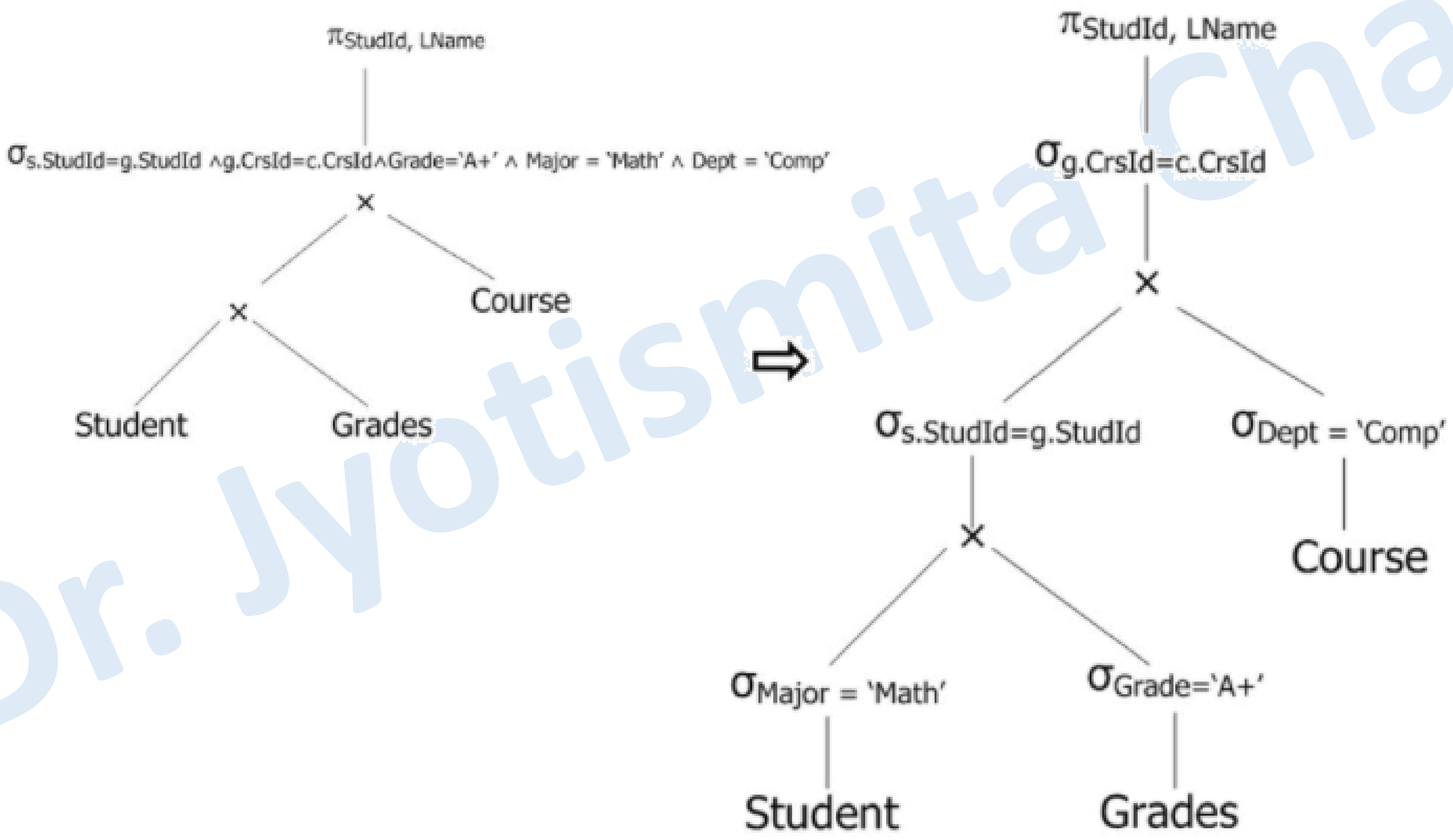
Heuristic Query optimization

Relational Algebra:

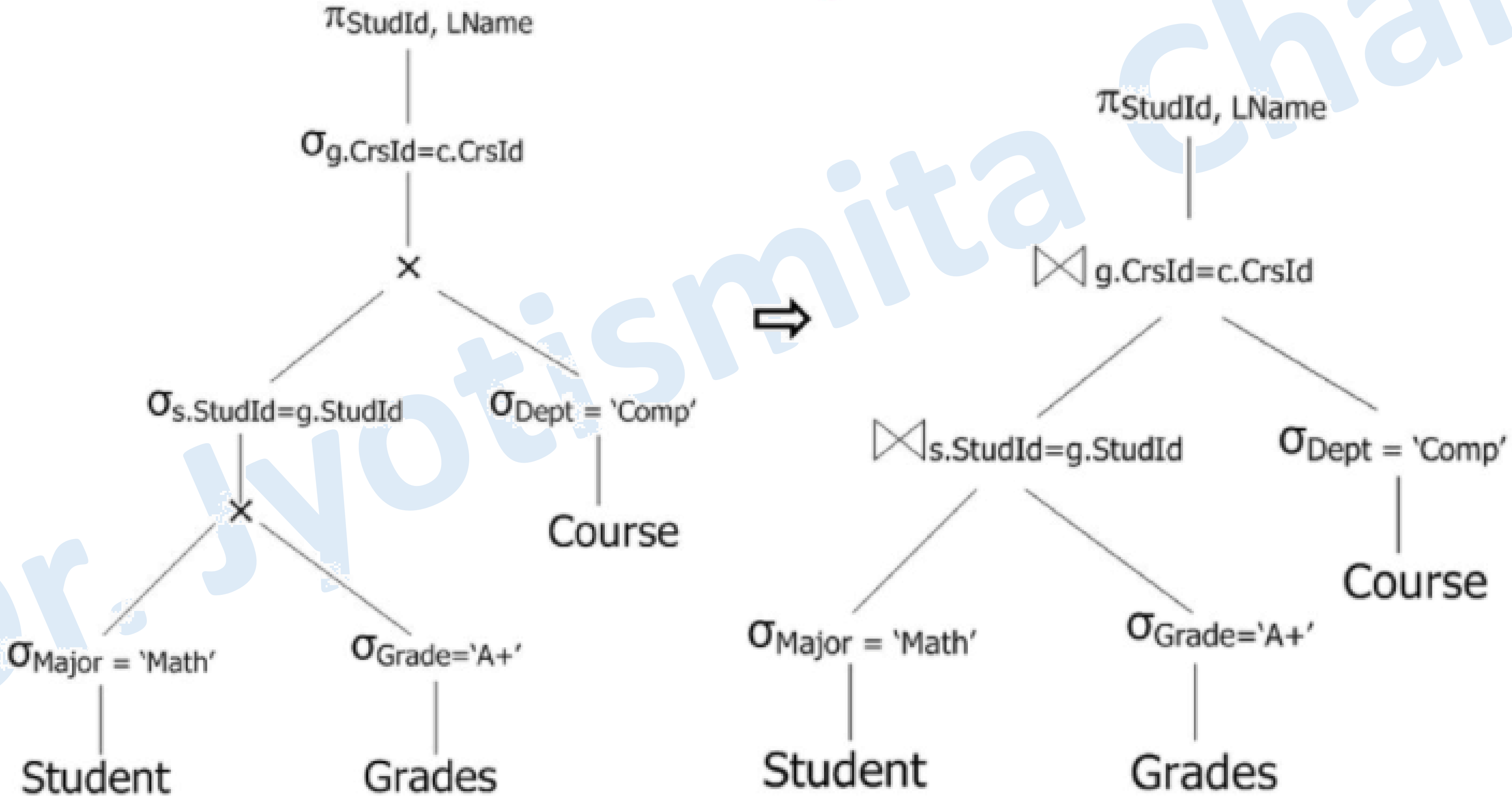
$$\pi_{StudId,\ LName}\ (\sigma_{s.StudId=g.StudId\ \wedge\ g.CrsId=c.CrsId\ \wedge\ Grade='A+'\ \wedge\ Major='Math'\ \wedge\ Dept='Comp'}\ (Course \times (Student \times Grades)))$$



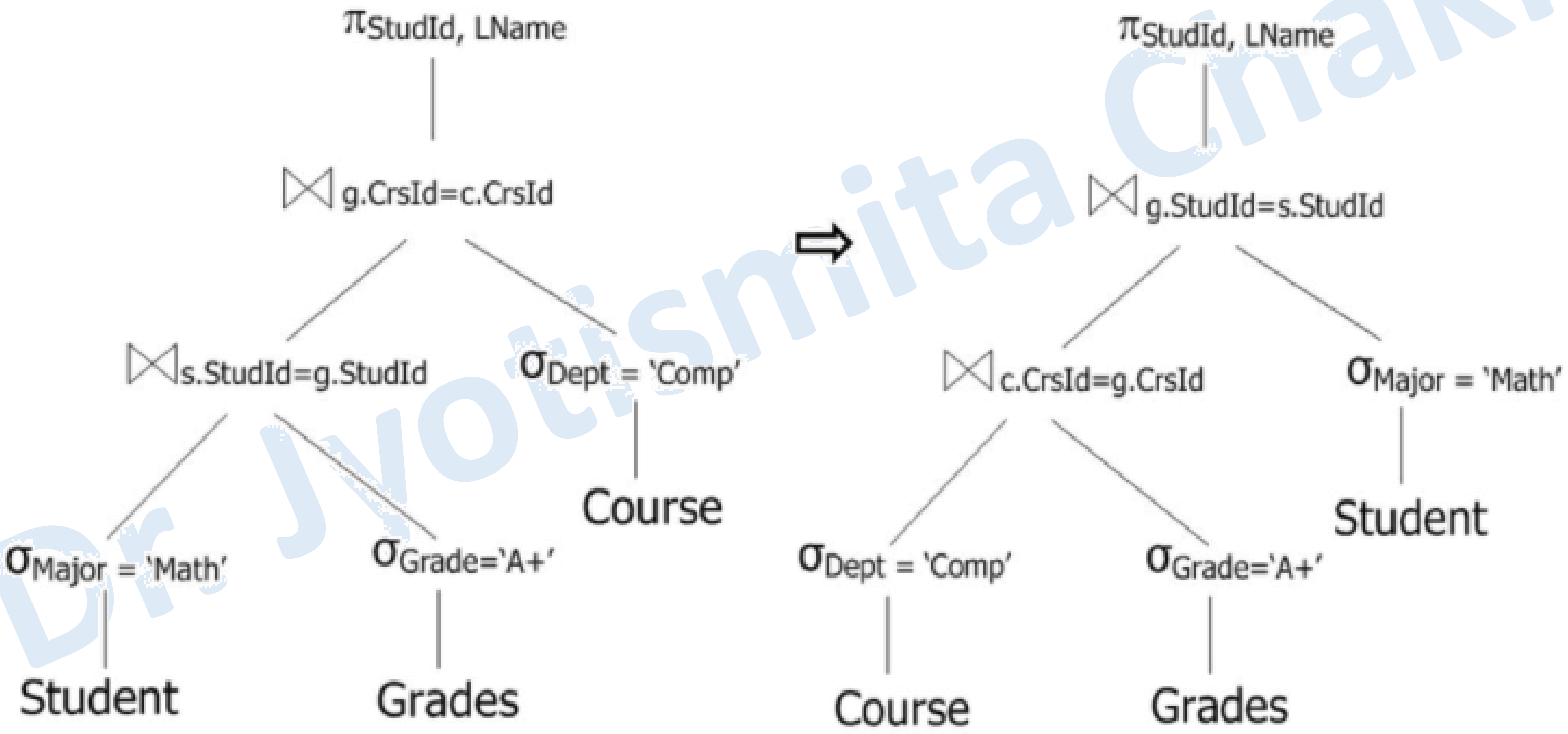
Heuristic Query optimization



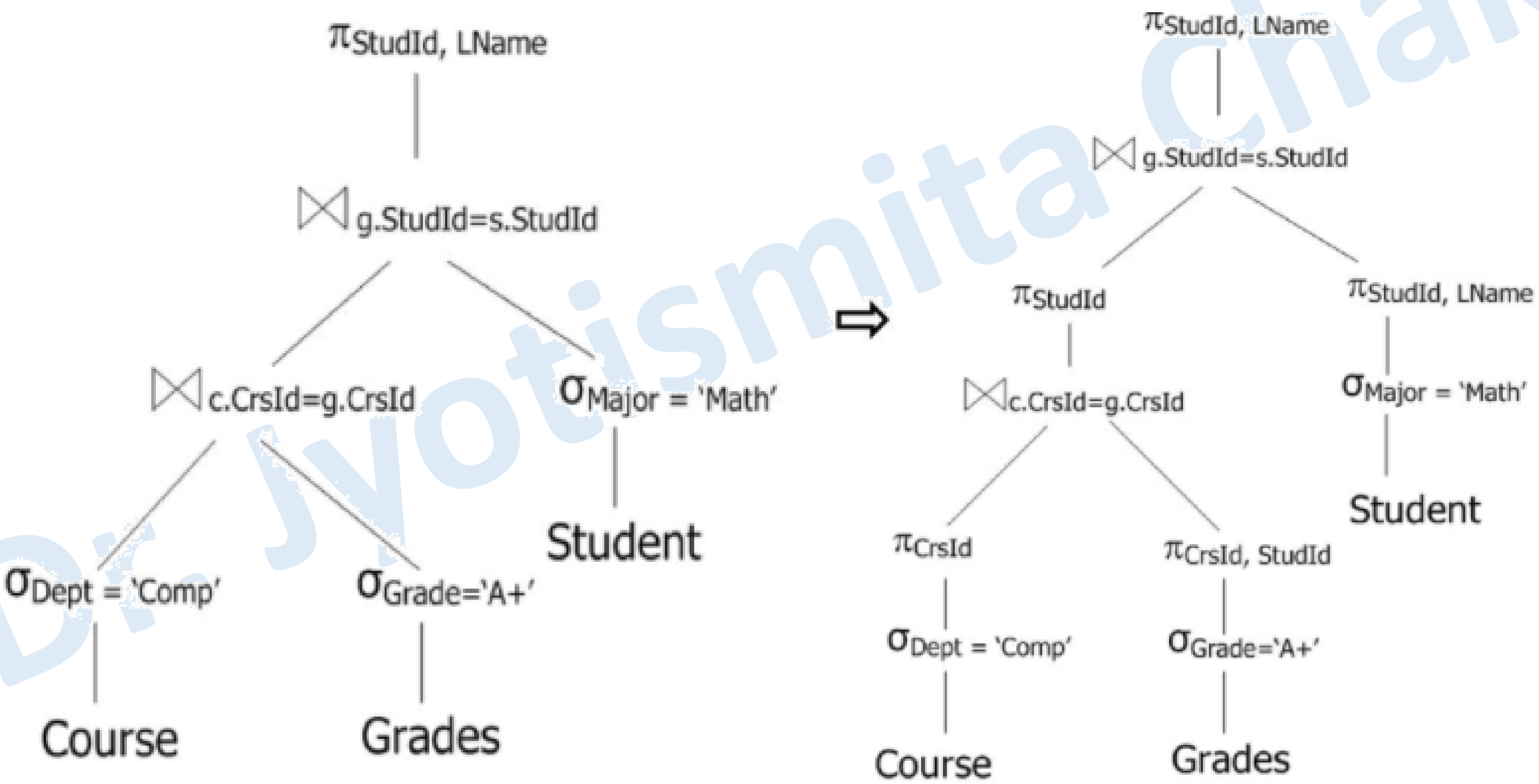
Heuristic Query optimization



Heuristic Query optimization



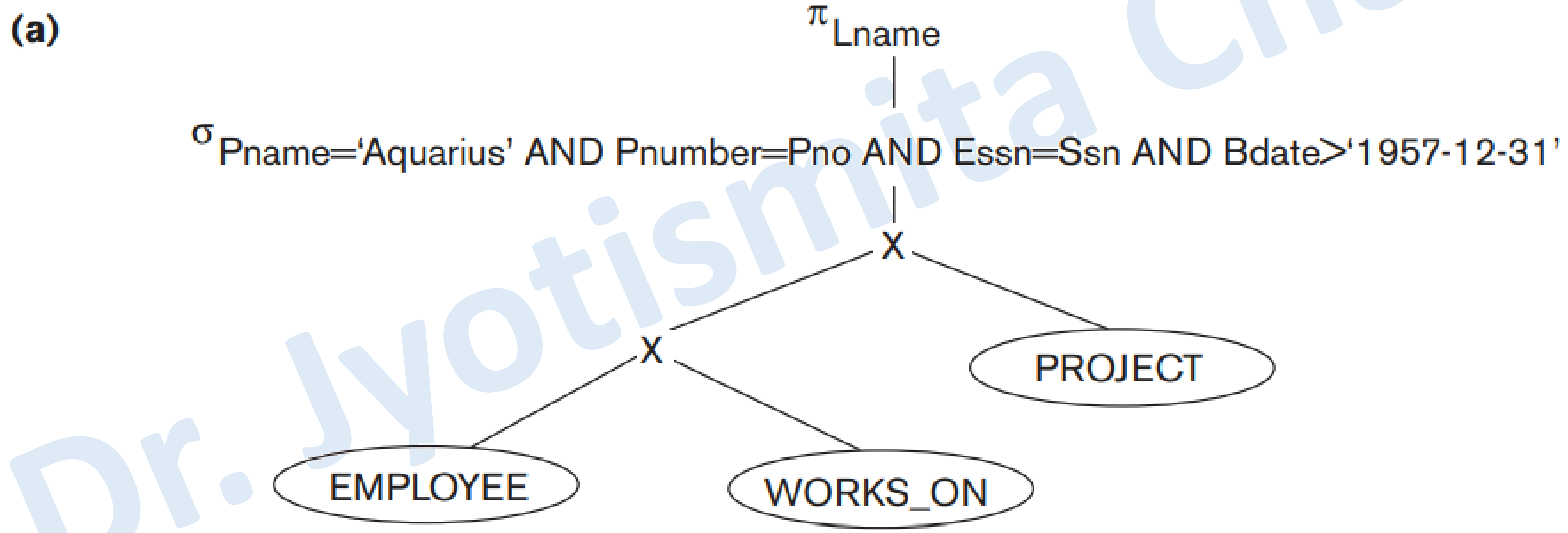
Heuristic Query optimization



Heuristic Query optimization

- **SELECT** E.Lname **FROM** EMPLOYEE E, WORKS_ON W, PROJECT P **WHERE** P.Pname='Aquarius' **AND** P.Pnumber=W.Pno **AND** E.Essn=W.Ssn **AND** E.Bdate > '1957-12-31'.
- Translation into RA:
  - $\pi_{Lname}\ (\sigma_{Pname='Aquarius' \wedge Pnumber=Pno \wedge Essn=W.Ssn \wedge Bdate > '1957-12-31'})\ (\rho_E(EMPLOYEE) \times \rho_W(WORKS\_ON) \times \rho_P(PROJECT))$
- The initial query tree for Q is shown in Figure (a).
- Executing this tree directly first creates a very large file containing the CARTESIAN PRODUCT of the entire EMPLOYEE, WORKS_ON, and PROJECT files.
- That is why the initial query tree is never executed, but is transformed into another equivalent tree that is efficient to execute.

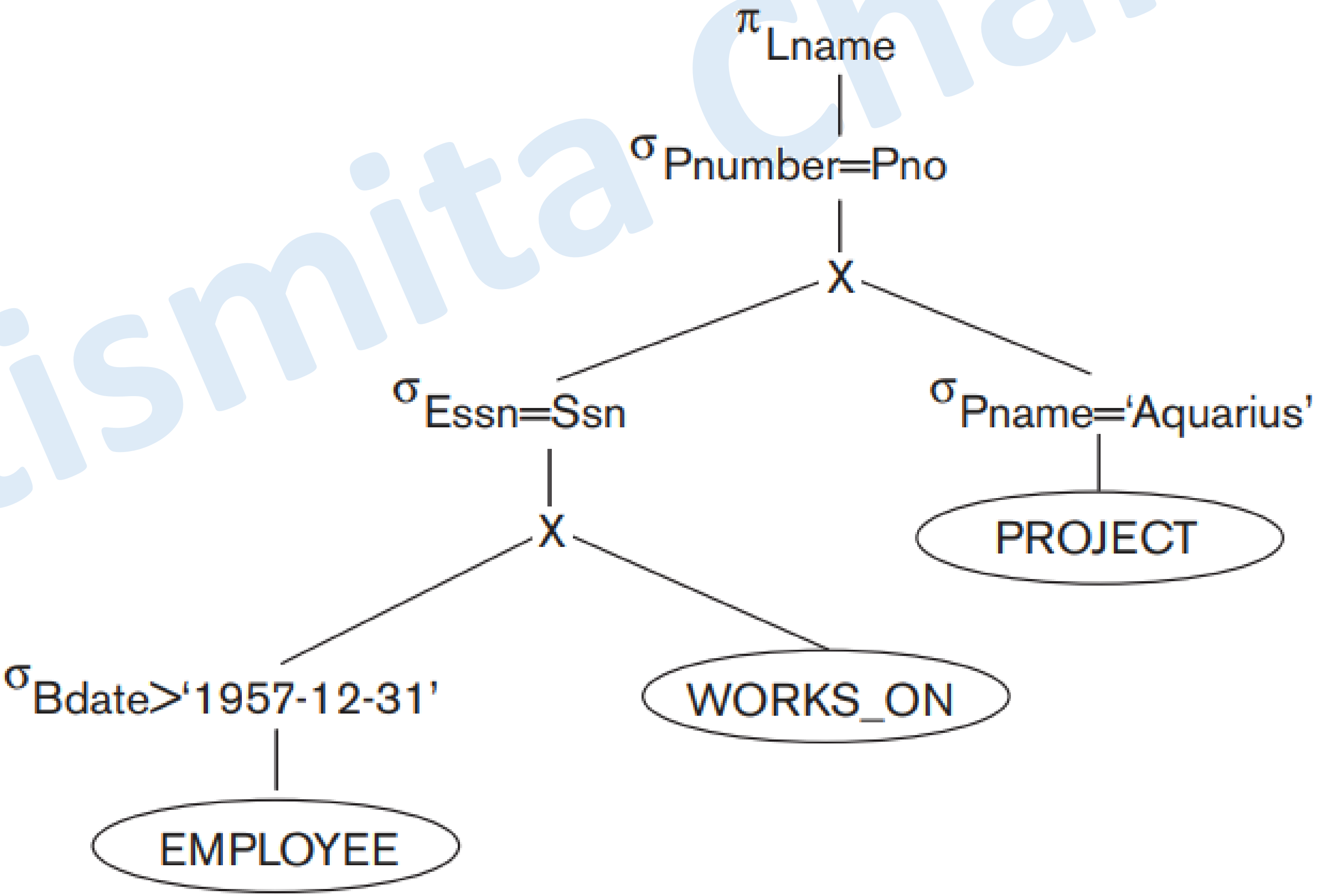
Heuristic Query optimization



Heuristic Query optimization

(b)

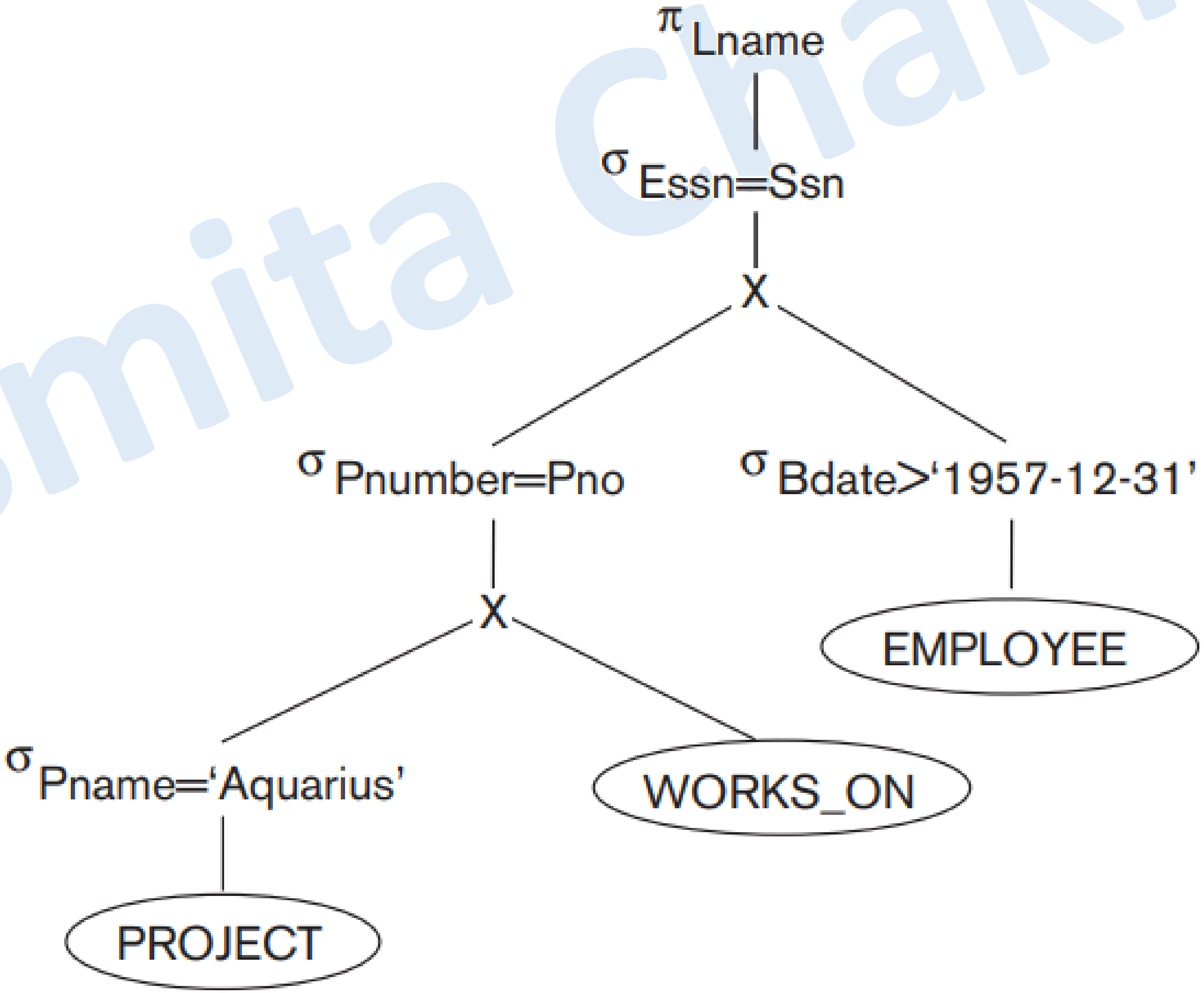
- Figure (b) shows an improved query tree that first applies the SELECT operations to reduce the number of tuples that appear in the CARTESIAN PRODUCT.



Heuristic Query optimization

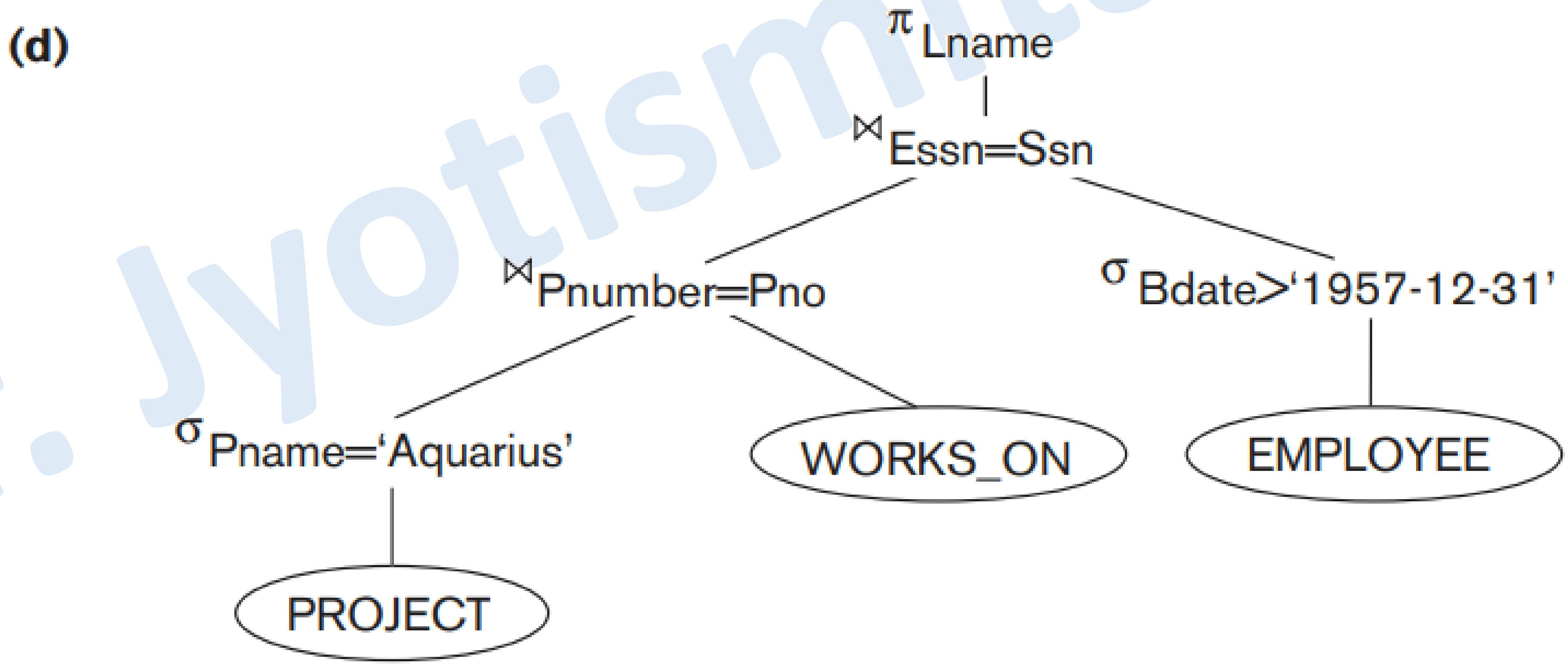
(c)

- A further improvement is achieved by switching the positions of the EMPLOYEE and PROJECT relations in the tree, as shown in Figure (c).
- This uses the information that Pnumber is a key attribute of the PROJECT relation, and hence the SELECT operation on the PROJECT relation will retrieve a single record only.



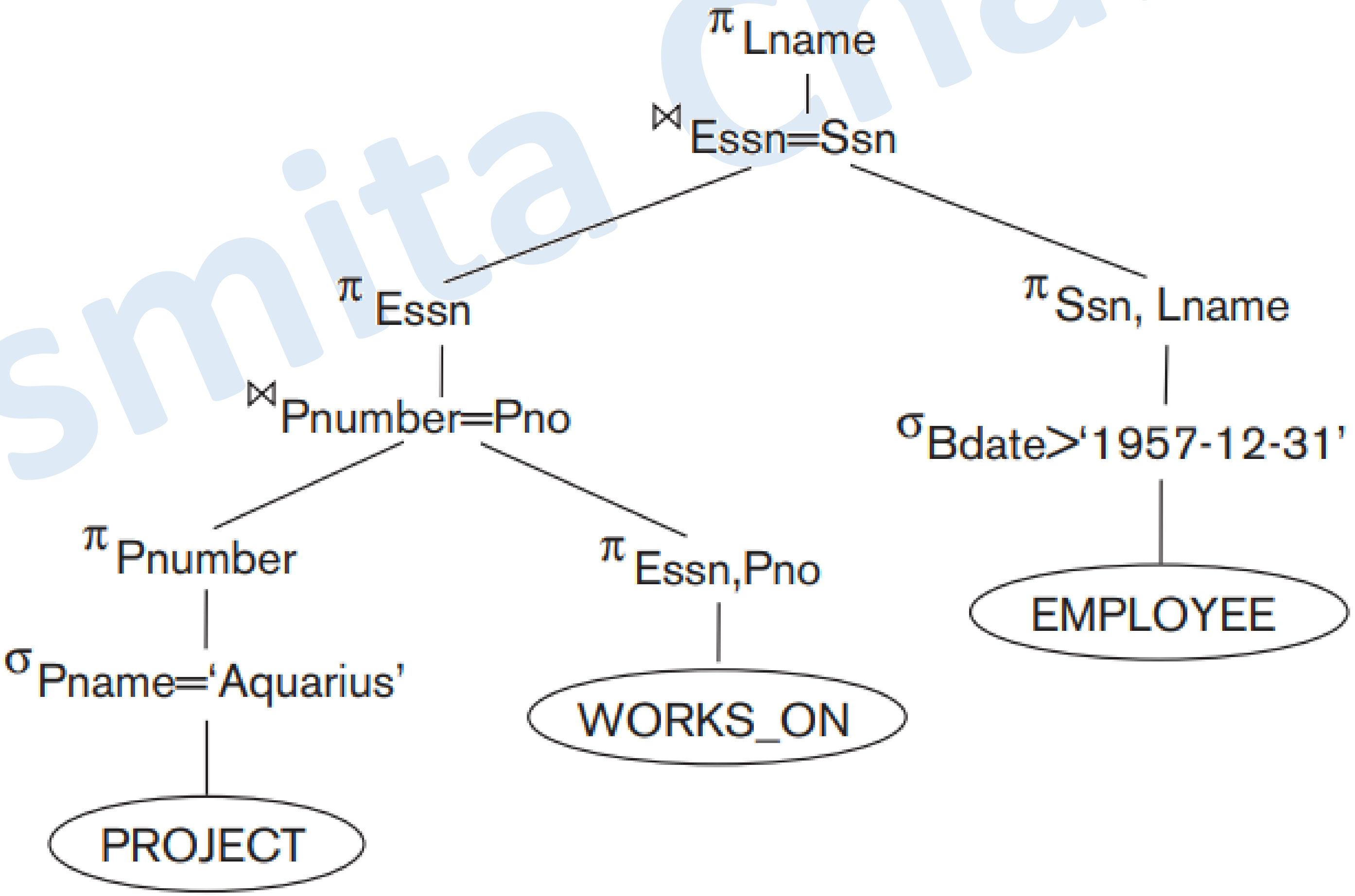
Heuristic Query optimization

- We can further improve the query tree by replacing any CARTESIAN PRODUCT operation that is followed by a join condition as a selection with a JOIN operation, as shown in Figure (d).



Heuristic Query optimization

(e)



- Another improvement is to keep only the attributes needed by subsequent operations in the intermediate relations, by including PROJECT (π) operations as early as possible in the query tree, as shown in Figure (e).
- This reduces the attributes (columns) of the intermediate relations, whereas the SELECT operations reduce the number of tuples (records).

Cost based query optimization

- A query optimizer does not depend solely on heuristic rules or query transformations; it also estimates and compares the costs of executing a query using different execution strategies and algorithms, and it then chooses the strategy with the lowest cost estimate.
- For this approach to work, accurate cost estimates are required so that different strategies can be compared fairly and realistically.
- In addition, the optimizer must limit the number of execution strategies to be considered; otherwise, too much time will be spent making cost estimates for the many possible execution strategies.
- This approach is generally referred to as **cost-based query optimization**.

Cost based query optimization

- It uses traditional optimization techniques that search the solution space to a problem for a solution that minimizes an objective (cost) function.
- The cost functions used in query optimization are estimates and not exact cost functions, so the optimization may select a query execution strategy that is not the optimal (absolute best) one.
- The query can use different paths based on indexes, constraints, sorting methods etc.
- This method mainly uses the statistics like record size, number of records, number of records per block, number of blocks, table size, whether whole table fits in a block, organization of tables, uniqueness of column values, size of columns etc.

Cost based query optimization: Cost Components for Query Execution

- **Access cost to secondary storage**. This is the cost of transferring (reading and writing) data blocks between secondary disk storage and main memory buffers. This is also known as disk I/O (input/output) cost.
- **Disk storage cost.** This is the cost of storing on disk any intermediate files that are generated by an execution strategy for the query.
- **Computation cost.** This is the cost of performing in-memory operations on the records within the data buffers during query execution. Such operations include searching for and sorting records, merging records for a join or a sort operation, and performing computations on field values. This is also known as CPU (central processing unit) cost.
- **Memory usage cost.** This is the cost pertaining to the number of main memory buffers needed during query execution.
- **Communication cost.** This is the cost of shipping the query and its results from the database site to the site or terminal where the query originated.

Cost based query optimization: Catalog Information

- To estimate the costs of various execution strategies, we must keep track of any information that is needed for the cost functions.
- This information may be stored in the DBMS catalog, where it is accessed by the query optimizer.
- First, we must know the size of each file.
- For a file whose records are all of the same type, the **number of records (tuples) (r)**, the (average) **record size (R)**, and the **number of file blocks (b)** (or close estimates of them) are needed.
- The **blocking factor (bfr)** for the file may also be needed.

Cost based query optimization: Catalog Information

- The **number of levels (x)** of each multilevel index (primary, secondary, or clustering) is needed for cost functions that estimate the number of block accesses that occur during query execution.
- In some cost functions the **number of first-level index blocks (b_{I1})** is needed.
- Another important parameter is the **number of distinct values NDV (A, R)** of an attribute in relation R and the **attribute selectivity (sl)**, which is the fraction of records satisfying an equality condition on the attribute.
- This allows estimation of the **selection cardinality (s)** of an attribute, which is the average number of records that will satisfy an equality selection condition on that attribute.

Cost based query optimization: Cost Functions for SELECT Operation

- The following notation is used in the formulas hereafter:
  - C_{Si}: Cost for method S_i in block accesses
  - r_X: Number of records (tuples) in a relation X
  - b_X: Number of blocks occupied by relation X (also referred to as b)
  - bfr_X: Blocking factor (i.e., number of records per block) in relation X
  - $sl_P(X)$: Selectivity of tuples in X for a given condition P
  - s_A: Selection cardinality of the attribute being selected
  - x_A: Number of levels of the index for attribute A
  - b_{I1A}: Number of first-level blocks of the index on attribute A
  - NDV (A, X): Number of distinct values of attribute A in relation X

Cost based query optimization: Cost Functions for SELECT Operation: Projection

- X(A, B, C) is a relation with A and B integers of 4 bytes each; C a string of 100 bytes. Tuple headers are 12 bytes. Blocks are 1024 bytes and have headers of 24 bytes. $r_X = 10000$ and $b_X = 1250$. How many blocks do we need to store $\pi_{A,B}(X)$?
- Resulting records need to record the header + A-field + B-field. The size of these records is hence 12 + 4 + 4 = 20 bytes. We can hence store (1024−24)/20 = 50 tuples in one block. Thus $b(\pi_{A,B}(X)) = r(\pi_{A,B}(X))/50 = 10000/50 = 200$ blocks.

Cost based query optimization: Cost Functions for SELECT Operation: Selection

- $\sigma_{A=c}(X)$ with c a constant: $sl_{A=c}(X) = 1/NDV(A, X)$.
  - Let, X(A, B, C) is a relation. $r_X = 10000$. NDV(A, X) = 50.
  - Then $r(\sigma_{A=10}(X))$ is estimated by: $r(\sigma_{A=10}(X)) = r_X \times 1/NDV\ (A, X) = 10000/50 = 200$.
- $\sigma_{A<C}(X)$: $sl_{A<C}(X) = ½$ or $sl_{A<C}(X) = 1/3$
  - X(A, B, C) is a relation. $r_X = 10000$.
  - Then $r(\sigma_{B<10}(x)) = r_X \times 1/3 = 10000 \times 1/3 = 3334$
- $\sigma_{A\neq c}(X)$: $sl_{A\neq c}(X) = [NDV\ (A, X) - 1]/\ NDV\ (A, X)$
- $\sigma_{P1\ AND\ P2}(X)$: $sl_{P1\ AND\ P2}(X) = sl_{P1}(X) \times sl_{P2}(X)$
  - X(A, B, C) is a relation. $r_X = 10000$. NDV(A, X) = 50.
  - Then we estimate $r(\sigma_{A=10\ AND\ B<10}(X)) = r_X \times sl_{A=10} \times sl_{B<10} = r_X \times [1/\ NDV(A, X) \times 1/3] = 67$

Cost based query optimization: Cost Functions for SELECT Operation: Selection

Selection conditions	**Computation**
Non-equality condition or Equality on a non-key: **Full table scan**	*b*
Equality condition on a key in unordered data: **Linear search**	*b/2*
Equality condition on a key in ordered data: **Binary Search**:	$\log_2(b)$
Equality condition on a key in ordered data using Primary Index:	x + 1
Equality condition on a non-key or Non-equality condition on a primary key Retrieve multiple records using an index: (assume 1/2 of the records match the condition)	x + (b/2)
Equality condition on a non-key using a clustering index:	x + (s/bfr)
Equality condition on a non-key using secondary index: because in the worst case, each record may reside on a different block.	x + s

Cost based query optimization: Cost Functions for SELECT Operation: Selection

Description	Computation
EMPLOYEE table has 10,000 records with a blocking factor of 5.	$r_E = 10{,}000$ $bfr_E = 5$ $b_E = 2000$
Assume a Secondary index on the key attribute EMPID	$x_{EMPID} = 4$ $s_{EMPID} = 1$
Assume a Secondary index on non-key attribute DNO	$x_{DNO} = 2$ $b_{11\ DNO} = 4.$
There are 125 distinct values of DNO that starts from 1 and ends at 125	$d_{DNO} = 125$
The selection cardinality of DNO is	$s_{DNO} = (r / d_{DNO}) = 80$
There are 200 distinct values of SALARY with Minimum salary of 50,000 and Maximum salary of 175,000	$s_{SALARY} = (r / d_{SALARY}) = 50$

Cost based query optimization: Cost Functions for SELECT Operation: Selection

- SELECT * FROM employee WHERE EMPID = 123456789

Description	Computation
1. Cost of a Full table Scan	b_E = 2000 blocks
2. Average Cost (no index) Linear search	b / 2 = 1,000 blocks
3. Cost when using a secondary index on EMPID	x + 1 = 4 + 1 = 5 blocks

- SELECT * FROM employee WHERE DNO = 5

Description	Computation
1. Cost of a Full table Scan	b_E = 2000 blocks
2. Rough estimate using the secondary index	$x_{DNO} + s_{DNO}$ = 2 + 80 = 82 blocks

Cost based query optimization: Cost Functions for SELECT Operation: Selection

SELECT * FROM employee WHERE DNO > 100

Variable	Computation
Selection Cardinality for non-equality (Assuming uniform distribution of distinct values)	For inequality ($A > c$): $S_A(R) = nTuples(R) * ((max_A(R) – c) / (max_A(R) – min_A(R))$ For inequality ($A < c$): $S_A(R) = nTuples(R) * ((c – min_A(R)) / (max_A(R) – min_A(R))$

Description	Computation
1. Cost of a Full table Scan	b_E = 2000 blocks
2. Rough estimate using the secondary index (assuming target records are in different blocks) Recall for inequality ($A > c$): $S_A(R) = nTuples(R) * ((max_A(R) – c) / (max_A(R) – min_A(R))$	$max_{DNO}(E) = 125$, $min_{DNO}(E) = 1$, $c = 100$ 10000*((125-100) / (125-1)) = Approx 2000 blocks

Cost based query optimization: Cost Functions for SELECT Operation: Selection

- SELECT * FROM employee WHERE SALARY > 130000 AND DNO = 4
- Plan 1:
  1. TEMP <- $\sigma_{DNO = 4}$ (E)
  2. $\sigma_{SALARY > 130000}$ (TEMP)
- We can assume nTuples(TEMP) = $S_{DNO}(E) = (r / d_{DNO}) = 80$
- Thus we can store nBlocks(TEMP) = nTuples(TEMP) / bfr_{TEMP} = 80/5 = 16

Description	**Computation**
1. Cost of a Full table scan + writing TEMP + full table scan on TEMP	$b_E + b_{TEMP} + b_{TEMP}$ = 2000 + 16 + 16 = 2032 blocks
2. Rough estimate using the secondary index followed by writing TEMP then a full table scan of TEMP	$(x_{DNO} + s_{DNO}) + b_{TEMP} + b_{TEMP}$ = (2 + 80) + (80/5) + (80/5) = 114 blocks

Cost based query optimization: Cost Functions for SELECT Operation: Selection

- Plan 2:
  1. TEMP <- $\sigma_{\text{SALARY} > 130000}$ (E)
  2. $\sigma_{\text{DNO} = 4}$ (TEMP)
- Since $\text{MIN}_{\text{SALARY}}$(E) = 50,000 and $\text{MAX}_{\text{SALARY}}$(E) = 175,000.
- Then nTuples(TEMP) = S_{SALARY}(E) = nTuples(E) * ($\text{MAX}_{\text{SALARY}}$ - c) / ($\text{MAXS}_{\text{ALARY}}$ - $\text{MIN}_{\text{SALARY}}$) = 10,000 * (175,000 – 130,000) / (175,000 – 50,000) = 3,600.
- Thus we can store nBlocks(TEMP) = nTuples(TEMP) / bfr_{TEMP} = 3600/5 = 720

Description	**Computation**
1. Cost of a Full table scan + writing TEMP + full table scan on TEMP	$b_E + b_{TEMP} + b_{TEMP}$ = 2000 + 720 + 720 = 3,440 blocks