

Database Management System
Prof. Partha Pratim Das
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture - 39
Query Processing and Optimization/2: Optimization

Welcome to module 39 of database management systems, we have been discussing about query processing and optimization, in the last module.

(Refer Slide Time: 00:25)



The slide is titled "Module Recap" in red text. It features a list of six topics, each preceded by a red square bullet point. The topics are: Overview of Query Processing, Measures of Query Cost, Selection Operation, Sorting, Join Operation, and Other Operations. The slide also includes a small image of a sailboat in the top left corner, a vertical text on the left side identifying the instructor as Prof. P. P. Das, and a footer with the text "Database System Concepts - 6th Edition", "39.2", and "©Silberschatz, Korth and Sudarshan".

We talked about the basic issues of query processing, what is the overview and the measures of query cost that would be used in terms of disk seek time and disk access time the read write time and we agreed we assume that we will ignore the CPU and other time for the time being. And then we took a look into how the certain SQL queries like the selection, the sorting which is required for other operations, different join operations and other aggregation and those kind of operations can be processed in a structured way.

(Refer Slide Time: 01:10)

PPD

Module Objectives

- To understand the basic issues for optimizing queries
- To understand how transformation of Relational Expressions can create alternates for optimization

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 39.3 ©Silberschatz, Korth and Sudarshan

In view of this in this module we would like to understand the basic issues of optimizing queries. So, that the same query as we have seen little bit can be performed executed in multiple ways and we would like to choose the one which is the least cost possibly estimated cost should be the least.

So, we would need to understand 2 aspects, one is given a query how do I generate alternate queries that is in terms of the relational expression how a particular expression can be transformed into equivalent expressions and then we choose from these equivalence expressions for optimization. So, these are the 2 topics to discuss.

(Refer Slide Time: 01:57)

Introduction

- Alternative ways of evaluating a given query
 - Equivalent expressions
 - Different algorithms for each operation

```
course(course_id, title, dept_name, credits)
instructor(I.D, name, dept_name, salary)
teaches(I.D, course_id, sec_id, semester, year)
```

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 38.6 ©Silberschatz, Korth and Sudarshan

So, to introduce on the query optimization let us take a simple example. So, I am referring to the university database that we have discussed earlier, it has 3 relations as listed above and using that we want to do the perform the query where we join instructor teaches and course these 3 relations and do a selection on that based on department name. So, this will give us naturally the instructors who are in teaching some course in the instructor is in music department and is teaching some course there and we want the name and title of these. So, we want the name of the instruction instructor and the title of the course that the person is teaching.

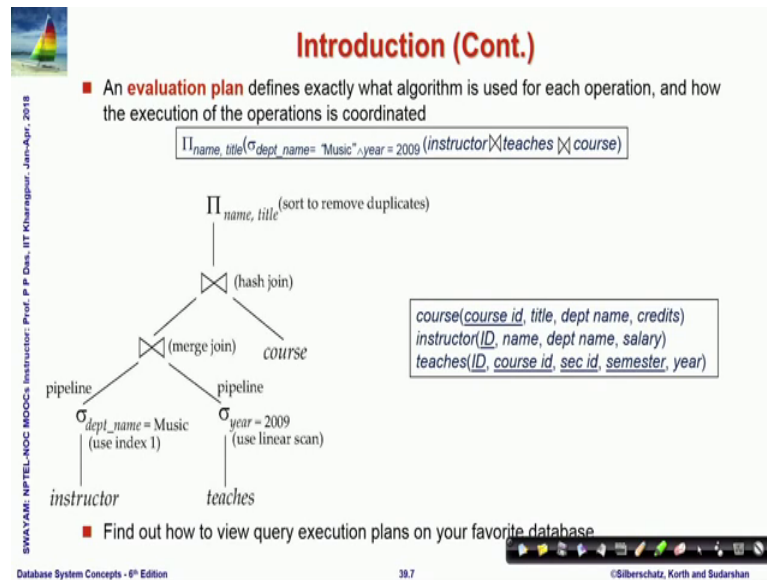
So, if we write the query in equivalent relational form this is what we will get to see and this is basically is a first join happening here then the next join happening, find next the selection and finally, the projection happening here which will give us the result of this query. Now what we observe is it is possible that if you look at carefully the department name should be music.

So, if we look into these relations then we can easily figure out that instructor has the department name attribute. So, in this after this joint if a tuple has to qualify through this selection then the instructors in the instructor relation the department name of that join people must be music otherwise it will not get selected.

So, what if instead of doing the join and then doing the selection as we are doing here if we first do a selection on the instructor relation itself and then join it with the join of

teachers and courses and finally, do the projection we should actually get the same result. So, these are what is it is a simple example of what are equivalent relational expressions that we can make use of in terms of our query processing.

(Refer Slide Time: 04:20)



So, what given that this is another example we were showing. So, here the query is to find the teacher and instructor and the course name for back the instructor is from department of the music and the course he taught is in the year 2009. So, we want these and corresponding to the equivalent at SQL if we write the relational expression then the in relational algebra this is what it looks like. So, what we can eventually observe from this that again as we observed last time department name is an is an instructor.

So, if we are doing a final selection based on department name is equal to is music then it is possible that I can first filter the instructor relation with the department name being music that should not affect the result. At the same time the other select condition we have is year is 2000, which happens only in the teacher's relation.

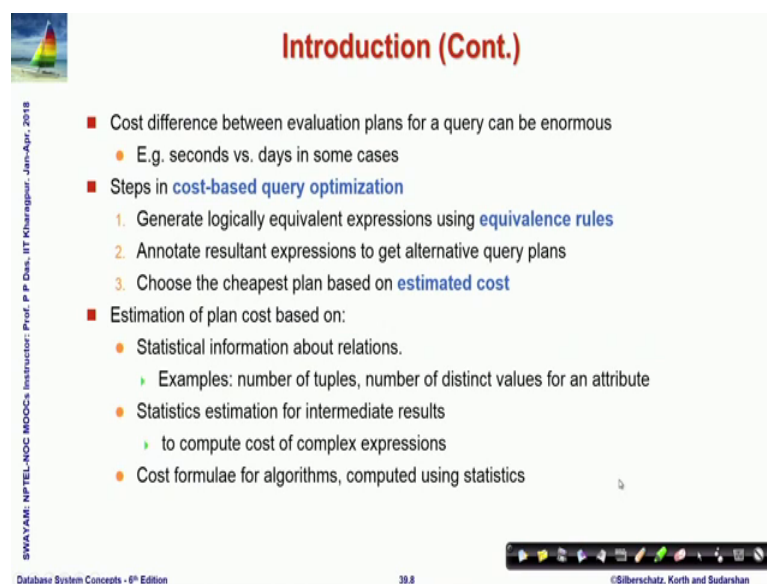
So, instead of actually filtering it after the join I can filter the teachers relation with year being 2009 and then use these 2 in terms of the join and finally, join that with the course and finally, do the projection. So, here what we show that along with this tree the parse tree of the query in relational algebra we are also trying to put some annotations to say how this particular query will be processed.

So, if we want to find out the tuples where the year is 2009 and there is no specific index on that or anything to for doing this selection as we have seen earlier the best way would be to use a linear scan. Whereas, if I am trying to do a selection with department name is equal to is music there will be a could be an index one the secondary index based on the department name. So, I will be able to use that index to find this.

So, these when we are doing this putting and that here we will use this index, here we will use linear scan these are what is called annotations which tell the query processing engine that how the query should actually be executed. Then they will be pipelined in the sense that this will be put one after the other actually these 2 can happen in parallel and then we will use a merge join to join these 2 then this merge that is joined result would be joined with course based on now the course is already indexed in course id and this joined relation of instructor and teachers will have id and course id on which they will have index.

So, we can use a hash join on that we did not discuss about merge join, hash join as processing steps in detail, but right now just assume that these are different ways of doing join which can make things efficient and these are the annotations which are generated in the process. And such a query tree such a query parse tree with the annotation is called an execution plan so, that or the evaluation plan which the query processing engine would be able to use to actually execute and find the results.

(Refer Slide Time: 07:58)



Introduction (Cont.)

- Cost difference between evaluation plans for a query can be enormous
 - E.g. seconds vs. days in some cases
- Steps in **cost-based query optimization**
 1. Generate logically equivalent expressions using **equivalence rules**
 2. Annotate resultant expressions to get alternative query plans
 3. Choose the cheapest plan based on **estimated cost**
- Estimation of plan cost based on:
 - Statistical information about relations.
 - ▶ Examples: number of tuples, number of distinct values for an attribute
 - Statistics estimation for intermediate results
 - ▶ to compute cost of complex expressions
 - Cost formulae for algorithms, computed using statistics

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 39.8 ©Silberschatz, Korth and Sudarshan

So, this is the basic approach so, for optimization what we need to do we need to find out that particular evaluation plan, that particular order of the evaluation and the use of algorithms, the use of indexes which will make the query possibly most efficient. And that cost difference could be really really huge in a real database it could vary between seconds in one way of doing the query or in terms of number of days if we do it in a non optimal way in a non optimized way.

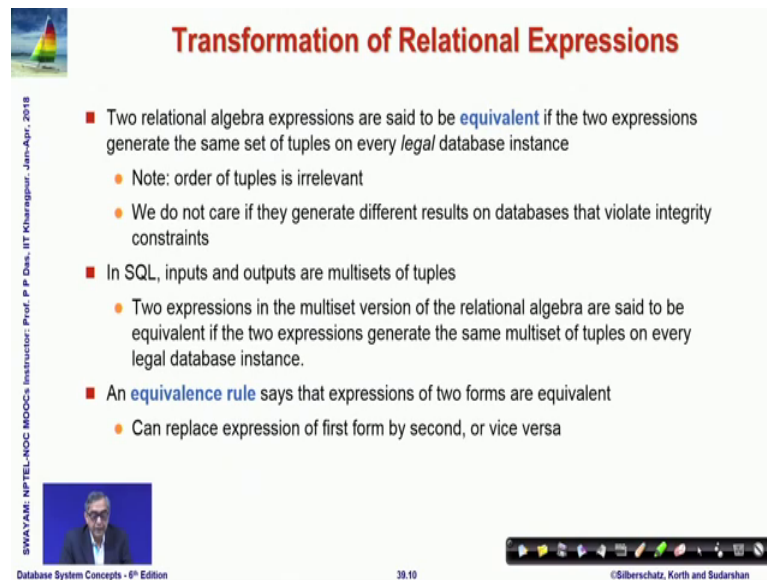
So, typical steps in this kind of query based optimization would be to first generate the candidates I am sorry first generate the candidates that is generate the equivalent expressions that is given a query I have one relational expression and we would like to generate equivalent expression using a set of equivalence rules we will see what these equivalence rules are.

So, that this equivalent queries expressions can any one of them can be actually executed and we then annotate them to with the result the resultant expression we annotate to get different query plans evaluation plans. And then we put the cost estimates based on the cost structure that say we had used in the other in the earlier module and from these alternate evaluation plans we will choose the one that will have the least estimated cost.

So, the cost can be based on as we had seen earlier it could be based on number of tuples, number of distinct values frequently used attributes and so, on and we will use statistics for also intermediate results that if there are intermediate results to be stored we will make estimates of what would be the size of that result because it needs to fit into memory for optimal execution, the cost formula for different algorithms will also be used through statistics.

So, based on all these estimation which will have an estimated cost and based on that we will choose the particular evaluation plan which looks to be the best and that is the crux of the query optimization strategy. So, as we have seen the first step is to be able to generate alternate expressions equivalent expressions through transformations. So, we look through the relational algebra operators again.

(Refer Slide Time: 10:27)



Transformation of Relational Expressions

- Two relational algebra expressions are said to be **equivalent** if the two expressions generate the same set of tuples on every *legal* database instance
 - Note: order of tuples is irrelevant
 - We do not care if they generate different results on databases that violate integrity constraints
- In SQL, inputs and outputs are multisets of tuples
 - Two expressions in the multiset version of the relational algebra are said to be equivalent if the two expressions generate the same multiset of tuples on every legal database instance.
- An **equivalence rule** says that expressions of two forms are equivalent
 - Can replace expression of first form by second, or vice versa

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 9th Edition

39.10

©Silberschatz, Korth and Sudarshan

And check what are what is meant by equivalence of 2 relational expressions. So, 2 relational expressions are equivalent if they generate the same set of tuples for any instance of the database, it is not enough to just show that for one instance it gives the same result.

So, 2 expressions are equivalent for in if I take any legal instance of the database then it must be equivalent and in this process we can note that the order of tuples are really relevant that we have told repeatedly and also we would make sure that the results are same provided the database provided the relation satisfy all the integrity constraints. If they violate integrity constraints then it is a problem of the user then we the database really does not care if the 2 expressions will give equivalent or equal results. We also note that in SQL input output could be multisets so, the same thing has to be satisfied in terms of multisets. So, based on this we define an equivalence a set of equivalence rule that say that 2 expressions are equivalent so, you can use that rule to transform one expression by the other and vice versa.

(Refer Slide Time: 11:47)

Equivalence Rules

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections
$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$
2. Selection operations are commutative
$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$
3. Only the last in a sequence of projection operations is needed, the others can be omitted
$$\Pi_{I_1}(\Pi_{I_2}(\dots(\Pi_{I_n}(\sigma_{\theta_1}(E))\dots))) = \Pi_{I_1}(E)$$
4. Selections can be combined with Cartesian products and theta joins
 - a. $\sigma_{\theta_1}(E_1 \times E_2) = E_1 \bowtie_{\theta_1} E_2$
 - b. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

So, let us take a look into this expression so, most of these expressions are relatively easy to understand. They can be formally proved using the corresponding set theoretic condition of the relational expressions. So, for every relational expression we had a set theoretic condition that we had studied so, using those you can prove that we will not do the proof of these equivalence relations here, but it you should be able to do that very easily.

So, he said if we have a conjunctive selection of a relation based on 2 conditions theta 1 and theta 2, then we should be able to apply the selection first based on theta 2 and then based on theta 1 or actually vice versa because conjunction is commutative so, the selection operation is also commutative. So, this gives us 2 transformation rules and we might want to use any so, these all will give equivalent form.

So, applying these 2 rules we get 3 relational expressions which are all equivalent this, this and this are all equivalent, but if you actually think in terms of processing the query and the cost involved you will be able to figure out that naturally the cost of doing this may be relatively more involved because you have the relation and then on the whole relation you are applying the conditions, but here if you do for example, if you first do say first apply theta 2 certainly by that selection the relation would become much smaller. So, applying theta 1 on that would be easier or vice versa depending on whichever is a smaller set there could be other for example, if you have a sequence of

projections then naturally in that sequence the last projection that you have done is what is retained.

So, if we have a sequence of projections that is simply doing the last projection all other projections can actually be ignored. The selection can be combined with Cartesian product and theta join also that is taking a Cartesian product and then doing a selection is equivalent to doing a theta join this of course, is almost the definition. And if I have a join by theta join by theta 2 and then do a selection with theta 1 it is equivalent to doing a theta join with theta 1 conjunction theta 2.

So, these are all equivalent forms so, these are equivalent in the sense that left hand side and right hand side are interchangeable. So, it does not mean that the left hand side implies the right hand side or vice versa it means that either of them implies the other, they are really equivalent in that sense.

(Refer Slide Time: 14:33)

Equivalence Rules (Cont.)

5. Theta-join operations (and natural joins) are commutative

$$E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$$

6. (a) Natural join operations are associative:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

(b) Theta joins are associative in the following manner:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

where θ_2 involves attributes from only E_2 and E_3 .

SWAYAM: NPTEL-NOC MOOC's Instructor: Prof. P. Das, IIT Kharagpur, Jan-Apr, 2018

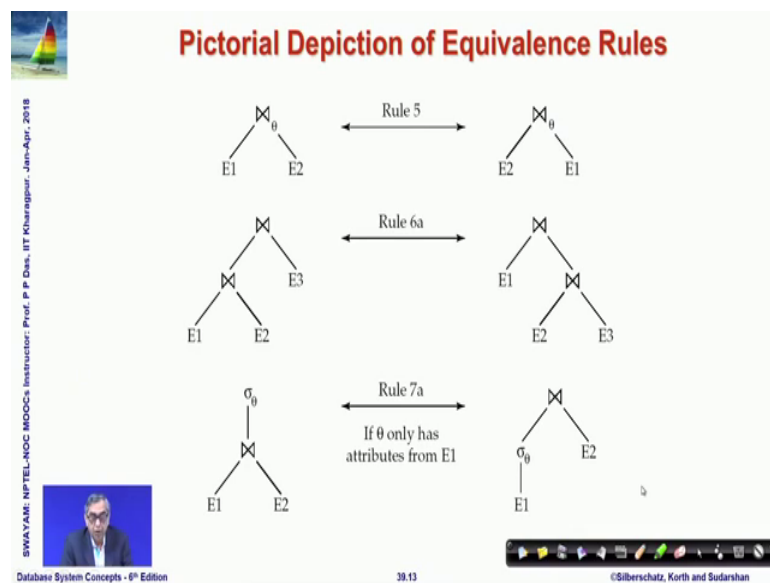
Database System Concepts - 9th Edition 39.12 ©Silberschatz, Korth and Sudarshan

Then theta join operation are natural or natural join operations are commutative, we can change interchange their relations.

So, this might impact for example, you have seen the algorithms of nested join and block join block nested join algorithms; obviously, depending on the size of the relations the cost of doing theta 1 E1 theta join E 2 or E1 natural join E2 and E2 natural join E1 may be different and we would choose the one which has a lesser cost.

Next set of transformation rules tell us that the join operation is associative which is obvious theta joins are associative in the in this specific manner also, that is if I do a theta join with condition theta 1 and then I do a theta join with condition theta 2 and theta 3 then we may be able to take out the condition theta 3 outside provided theta 2 the condition theta 2 uses only the attributes of theta E1 and E2 so, it should be possible to do that. Look here that on the left hand side that restriction is not there on the condition theta 2 it could also use attributes of E1, but if it does not then it is possible to simplify it in this manner and these are the equivalent rules that we have.

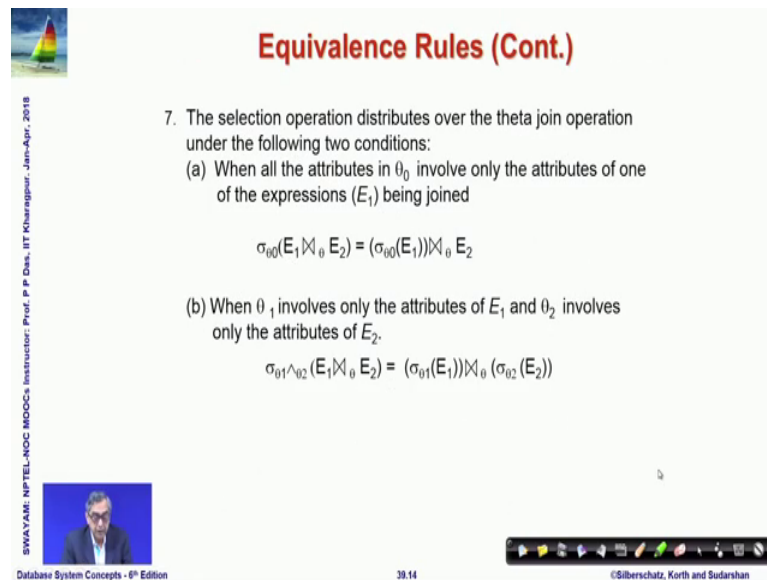
(Refer Slide Time: 16:01)



So, often we will draw the rules in this form so, just to explain you one so, this shows the associativity of join.

So, here what you are saying is first you do E1, E2 and with the result you join E3, here you are saying first you do E2, E3 and then you join with E1. So, this is basically associativity this basically is commutativity of theta join. So, we will often draw them in such forms of parse trees and show the equivalence that becomes easy to understand for example, in this case you are doing a join and then doing the selection and here you are doing it select early you are doing a select early on this relation E1 of course, you will be able to do this provided theta contents only attribute from E1, if theta contains attributes from both E1 and E2 this transformation will not be applicable this transformation will not be possible.

(Refer Slide Time: 17:00)



Equivalence Rules (Cont.)

7. The selection operation distributes over the theta join operation under the following two conditions:

(a) When all the attributes in θ_0 involve only the attributes of one of the expressions (E_1) being joined

$$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$$

(b) When θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2 .

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$$

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P.P. Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 8th Edition

39.14

©Silberschatz, Korth and Sudarshan

So, these are some more of the transformation rules the selection operation distributes over theta join operation. So, I can see here that there is a theta join and then I am doing a selection. So, I can first do the selection and then do the theta join of course, for the selection it must involve only the attributes of E_1 , otherwise this will not be valid.

And similarly, another distribution rule is shown here where you are doing a selection on conjunction on theta 1, theta 2 on a theta join by theta then you should be able to actually distribute based on the condition theta 1 and theta 2, if theta 1 involves only the attributes of E_1 and theta 2 involves only the attributes of E_2 . These are these are pretty straightforward rules if you think about the corresponding set theoretic reason.

(Refer Slide Time: 17:58)

Equivalence Rules (Cont.)

8. The projection operation distributes over the theta join operation as follows:

(a) if θ involves only attributes from $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\theta} E_2) = (\Pi_{L_1}(E_1)) \bowtie_{\theta} (\Pi_{L_2}(E_2))$$

(b) Consider a join $E_1 \bowtie_{\theta} E_2$.

- Let L_1 and L_2 be sets of attributes from E_1 and E_2 , respectively
- Let L_3 be attributes of E_1 that are involved in join condition θ , but are not in $L_1 \cup L_2$, and
- Let L_4 be attributes of E_2 that are involved in join condition θ , but are not in $L_1 \cup L_2$.

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\theta} E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_{\theta} (\Pi_{L_2 \cup L_4}(E_2)))$$

Database System Concepts - 8th Edition
©Silberschatz, Korth and Sudarshan

So, the other rules are will be in terms of projection so, you can have if you have union projection like this then you would be able to break it down over to do separate projections and their theta join and in case you have a theta join of E1, E2 based on theta and if L1 and L2 are the attributes of these 2 relations.

So, if L3 be the attributes of E1 that are involved in the join condition theta and L4 are water. So, you have the join condition theta. So, what you are saying that the attributes L3 of you are not involved here and attributes L4 of E2 are involved here, but they are not so, in terms of L1 union L2 so, then this kind of I should be able to distribute the projection in steps and make the results smaller.

(Refer Slide Time: 19:04)

Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative
 $E_1 \cup E_2 = E_2 \cup E_1$
 $E_1 \cap E_2 = E_2 \cap E_1$
■ (set difference is not commutative).

10. Set union and intersection are associative.
 $(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$
 $(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$

11. The selection operation distributes over \cup , \cap and $-$.
 $\sigma_{\theta}(E_1 - E_2) = \sigma_{\theta}(E_1) - \sigma_{\theta}(E_2)$
and similarly for \cup and \cap in place of $-$
Also: $\sigma_{\theta}(E_1 - E_2) = \sigma_{\theta}(E_1) - E_2$
and similarly for \cap in place of $-$, but not for \cup

12. The projection operation distributes over union
 $\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$

Database System Concepts - 6th Edition
©Silberschatz, Korth and Sudarshan

So, this is another possible transformation that now finally, the set theoretic operations have their normal set rules so, union and intersection are commutative, naturally set difference is not commutative, union intersection are associative as well. The selection operation distributes over union, intersection and set difference and the projection also distributes over union. So, you can see the exceptions here you can reason that out.

(Refer Slide Time: 19:35)

Exercise

- Create equivalence rules involving
 - The group by/aggregation operation
 - Left outer join operation

Database System Concepts - 6th Edition
©Silberschatz, Korth and Sudarshan

So, we have in short we have presented a set of different transformation rules by each which you can make equivalent expressions and between 2 equivalent expressions or

more equivalent expressions our objective will always be to choose the one whose evaluation plan will have a lesser cost. So, as an exercise I have left that you can create equivalence rules that involve group by or aggregation operations or different kinds of outer join, left outer join, right outer join and so, on so, please work those out.

(Refer Slide Time: 20:05)

Transformation Example: Pushing Selections

- Query: Find the names of all instructors in the Music department, along with the titles of the courses that they teach
- $\pi_{name, title}(\sigma_{dept_name = 'Music'}(instructor \bowtie (teaches \bowtie \pi_{course_id, title}(course))))$
- Transformation using rule 7a
- $\pi_{name, title}((\sigma_{dept_name = 'Music'}(instructor)) \bowtie (teaches \bowtie \pi_{course_id, title}(course)))$
- Performing the selection as early as possible reduces the size of the relation to be joined

Relations:

- course(course_id, title, dept_name, credits)
- instructor(ID, name, dept_name, salary)
- teaches(ID, course_id, sec_id, semester, year)


Let us move on to a couple of examples so, here is a query here the relations from the university database, here is a query find the names of all instructors in the music department along with the titles of the course they teach. So, a while ago we saw this so, this is what the query is in the relational algebra form and if you use the transformation rule of select early the rule 7a, whose transformation I have just shown here then you should be able to hear what you are doing is you are first doing a join of teaches and the projection of course. And then joining instructor with that and finally, doing the selection, but you should you would be able to do this select early because it involves the attribute department name which is the attribute of instructor alone here.

So, you should be able to first do this selection, mind you here courses also show that there is an attribute department name, but actually the courses is being used after projection. So, after projection in the projected relation there is no department name. So, department name is involved only the instructor.

So, I can first I can take this do early, I can do this selection early and take this out and then do the final join operation. So, in that process possibly the this will reduce the size

of the relation to be joined significantly because you do not naturally expect to be to have too many instructors in the music department. So, the instructor relation after the selection would become much smaller.

(Refer Slide Time: 21:52)

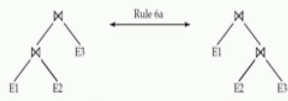


Example with Multiple Transformations

- Query: Find the names of all instructors in the Music department who have taught a course in 2009, along with the titles of the courses that they taught
 - $\Pi_{name, title}(\sigma_{dept_name = \text{Music} \wedge year = 2009}(\text{instructor} \bowtie (\text{teaches} \bowtie \Pi_{course_id, title}(\text{course}))))$
- Transformation using join associativity (Rule 6a):
 - $\Pi_{name, title}(\sigma_{dept_name = \text{Music} \wedge year = 2009}(((\text{instructor} \bowtie \text{teaches}) \bowtie \Pi_{course_id, title}(\text{course})))$
- Second form provides an opportunity to apply the "perform selections early" rule, resulting in the subexpression

$$\sigma_{dept_name = \text{Music}}(\text{instructor}) \bowtie \sigma_{year = 2009}(\text{teaches})$$

course(course_id, title, dept_name, credits)
 instructor(ID, name, dept_name, salary)
 teaches(ID, course_id, sec_id, semester, year)



Database System Concepts - 9th Edition
39.19
©Silberschatz, Korth and Sudarshan

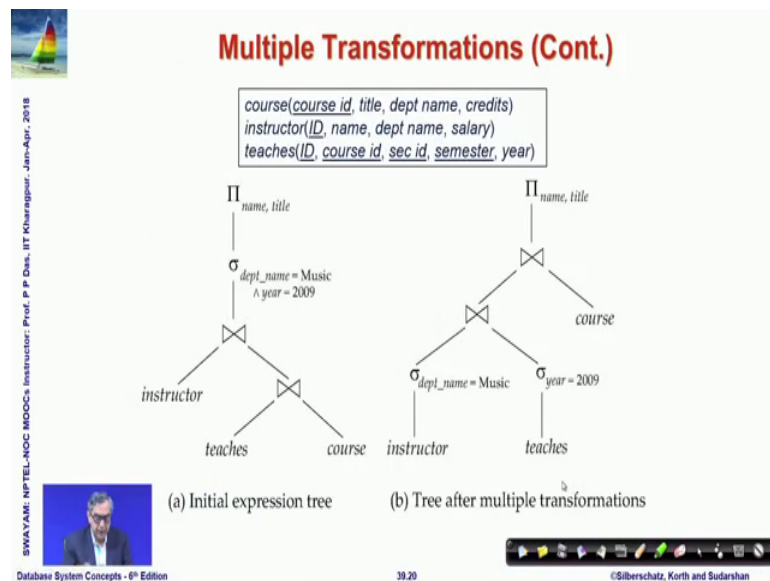
So, this is one example, this is another example query we are taking find the names of all instructors in the music department, we have taught a course in 2009 along with the titles of the course they taught. So, here is a the corresponding here is the corresponding relational query which you can convince yourself is indeed the same.

And then we can transform using the rule 6a which I have shown here which is basically the associativity of joint because there are 2 join relations and we are using the associativity of join to first join the here then second and third relations are joined first and then the first relation is joined with that. Here we are using associativity of joining the first and second and then using the third, now if you join first and second and then it is possible that the department name actually the department name is an instructor and the other condition is here which is in the teaches.

So, this department name is in the instruction here is in the teaches. So, I should be able to do select early, I should be able to select on the instructor based only on the department name being music and also select early on the teachers by using a year as 2009 and then do their joint.

So, naturally each one of these will become much smaller corresponding to the whole instructor or teaches relation therefore, their join will also be smaller. So, which means eventually this part this part of the query will become much smaller in size in the result and the consequent second join would be much smaller to perform. So, this will naturally give it whereas, if we had done all of these join earlier we would have used the whole of the teaches and the instructor relations and that would have been quite a lot of tuples would have been there.

(Refer Slide Time: 23:57)



So, these are different example so, this is a the same example being shown in terms of the 3 parts tree structure so, we can convince yourself by using the transformation rules that they are actually equivalent.

(Refer Slide Time: 24:12)

Transformation Example: Pushing Projections

- Consider: $\Pi_{name, title}(\sigma_{dept_name = \text{"Music"}}(instructor) \bowtie teaches) \bowtie \Pi_{course_id, title}(course)$
- When we compute $(\sigma_{dept_name = \text{"Music"}}(instructor) \bowtie teaches)$ we obtain a relation whose schema is: $(ID, name, dept_name, salary, course_id, sec_id, semester, year)$
- Push projections using equivalence rules 8a and 8b; eliminate unneeded attributes from intermediate results to get: $\Pi_{name, title}(\Pi_{name, course_id}(\sigma_{dept_name = \text{"Music"}}(instructor) \bowtie teaches)) \bowtie \Pi_{course_id, title}(course)$
- Performing the projection as early as possible reduces the size of the relation to be joined

course(course_id, title, dept name, credits)
instructor(ID, name, dept name, salary)
teaches(ID, course_id, sec_id, semester, year)

$$\Pi_{I_1 \cup I_2}(E_1 \bowtie E_2) = (\Pi_{I_1}(E_1)) \bowtie (\Pi_{I_2}(E_2)) \quad \text{8a}$$

$$\Pi_{I_1 \cup I_2}(E_1 \bowtie E_2) = \Pi_{I_1 \cup I_2}((\Pi_{I_1 \cup I_2}(E_1)) \bowtie (\Pi_{I_1 \cup I_2}(E_2))) \quad \text{8b}$$

And then certainly they give you a significant advantage and now this is an example which show that you can push the projection.

So, here is what you wanted to do and while we compute this we will get a relation of this form and we can push, we can use these rules rule 8a and 8b, this is rule 8a, this is rule 8b by this we can push the projections inside and make the relations smaller because now if you push the projection then you are actually cutting down on the on the number of columns. So, your relation becomes smaller that will make the with the projection this will reduce and when you project also there will be duplicates which will get removed so, in every way your relation becomes smaller in size and your subsequent join operations would be more efficient.

(Refer Slide Time: 25:02)

Join Ordering Example

- For all relations r_1, r_2 and r_3 ,
 $(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$
(Join Associativity)
- If $r_2 \bowtie r_3$ is quite large and $r_1 \bowtie r_2$ is small, we choose
 $(r_1 \bowtie r_2) \bowtie r_3$
so that we compute and store a smaller temporary relation

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Khargpur, Jan-Apr, 2018

You can also see these are examples where you can take care of the fact that you can reorder joining to get better results. For example, if I have to do a join of 3 relations like this, I could do either this first or this first. Now if I do this first then this is a temporary relation which I will need to maintain in memory and then join with r_1 . If I do this first then this will be a temporary relation and then I will join it with r_3 . So, if this is large enough then compared to this then I will be better off by doing this and will be able to have a more optimized execution of the query. So, here is an example of that again 2 joins.

(Refer Slide Time: 25:46)

Join Ordering Example (Cont.)

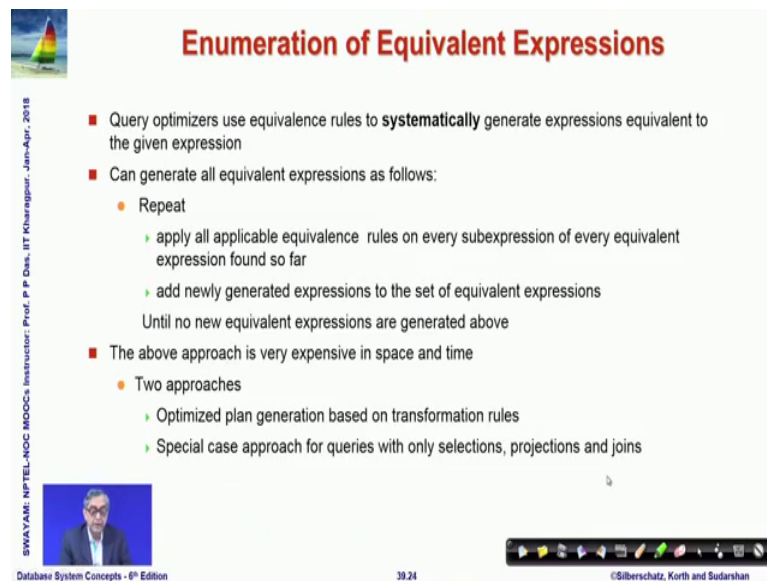
- Consider the expression
 $\Pi_{name, title}(\sigma_{dept_name = \text{Music}}(instructor) \bowtie teaches) \bowtie \Pi_{course_id, title}(course))$
- Could compute $teaches \bowtie \Pi_{course_id, title}(course)$ first, and join result with $\sigma_{dept_name = \text{Music}}(instructor)$
but the result of the first join is likely to be a large relation
- Only a small fraction of the university's instructors are likely to be from the Music department
 - it is better to compute
 $\sigma_{dept_name = \text{Music}}(instructor) \bowtie teaches$
first

$course(course_id, title, dept_name, credits)$
 $instructor(ID, name, dept_name, salary)$
 $teaches(ID, course_id, sec_id, semester, year)$

Database System Concepts - 6th Edition 39.23 ©Silberschatz, Korth and Sudarshan

So, what we are trying to show is when we are having this instead of actually are actually trying to compute this join first because we expect that this set to be much smaller, if this set is much smaller then naturally this joint would be much smaller and it is more likely to fit into the memory then if we had just done teaches and course id's which I expected to be large relations.

(Refer Slide Time: 26:20)



Enumeration of Equivalent Expressions

- Query optimizers use equivalence rules to **systematically** generate expressions equivalent to the given expression
- Can generate all equivalent expressions as follows:
 - Repeat
 - ▶ apply all applicable equivalence rules on every subexpression of every equivalent expression found so far
 - ▶ add newly generated expressions to the set of equivalent expressionsUntil no new equivalent expressions are generated above
- The above approach is very expensive in space and time
 - Two approaches
 - ▶ Optimized plan generation based on transformation rules
 - ▶ Special case approach for queries with only selections, projections and joins

SWAYAM: NPTEL-NOC MOOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 39.24 ©Silberschatz, Korth and Sudarshan

So, basically therefore, the strategy turns out that given a query you will have to generate it whole lot of equivalent expressions. So, the number of equivalent expressions could be really really large. So, we will have to systematically generate all these alternate equivalent expressions and apply by applying these transformation rules that we have just seen and we will have to continue till no new expression can be generated and then we have to evaluate each one of them based on their evaluation plan.

So, this could be very expensive because the number of alternates could be really really large. So, the optimize plan generation is also based on the transformation rules and we may have only different special kits approaches to take care of the common optimization plans that we might have.

(Refer Slide Time: 27:16)

Implementing Transformation Based Optimization

- Space requirements reduced by sharing common sub-expressions:
 - when E1 is generated from E2 by an equivalence rule, usually only the top level of the two are different, subtrees below are the same and can be shared using pointers
 - ▶ E.g. when applying join commutativity
- Same sub-expression may get generated multiple times
 - ▶ Detect duplicate sub-expressions and share one copy
- Time requirements are reduced by not generating all expressions
 - Dynamic programming

The diagram shows two join operations. The left join has two children, E1 and E2. The right join has two children, E2 and E1. Arrows point from the E1 child of the left join to the E1 child of the right join, and from the E2 child of the left join to the E2 child of the right join, illustrating shared subtrees.

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-April, 2018

Database System Concepts - 8th Edition

39.25

©Silberschatz, Korth and Sudarshan

Now, one way to do that is for example, if we are looking at say the join of 2 relations E1 and E2 here then; obviously, if we do certain transformations with this join that does not change the way E1 and E2 are actually evaluated.

So, if that be the case then in terms of generating the alternate we do not really need to keep or evaluate all of the expressions the whole of the expression in every alternate. We could actually do something like sorry we could actually do something like we could have a join and then instead of really replicating the whole of the evaluation of E1 and evaluation of E2, we can simply make pointers to the same sub trees which are called basically sub expression optimization.

If you have studied the expression optimization in compiler at any point of time you will understand this very well, this is the same strategy which is used here. So, if you can detect duplicate sub expressions then you can have only one copy and you can make the things more efficient to run.

So, the general strategy for doing this is a dynamic programming we would not be able to cover that in the current course, but just know that all these explosion of generating alternate and choosing the best one is usually handled in terms of dynamic programming which is a strategy to make sure that if we have solved a sub earlier then I do not need to solve that sub problem again we can just reuse that same earlier result and go ahead with that.

So, by this way we can common sub expressions we may only find the plan for a best plan for a common sub expression only once and then use it subsequently again.

(Refer Slide Time: 29:24)

Module Summary

- Understood the basic issues for optimizing queries
- For every relational expression, usually there are a number of equivalent expressions that can be created by simple transformations
- Final execution plan can be created by choose the estimated least cost expression from the alternates

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 39.26 ©Silberschatz, Korth and Sudarshan

So, in this module covered starting from the notions of query processing in the earlier module, we have discussed the basic issues of optimizing queries and we have shown that using a set of simple transformational rules you can convert a relational expression into a number of equivalent relational expressions. And then you can evaluate them based on the estimated cost that the model that you are using and choose the best one and dynamic programming is usually a good way of doing that.

So, in through the previous module and this one we have taken a very elementary look I should say, but this will give you some idea of how actually an SQL query is transformed into relational algebra parsed and translated into relational algebra and how equivalent expressions for that relational algebra expression is generated and evaluated.

And finally, an evaluation plan is met where specific choice is made for the different algorithms for doing different operations and that final plan which is which has the best cost is passed on to the query processing engine to query evaluation engine. And that then goes forward and does the b tree and disk operations in according to the plan and produces the best result in possibly the best shot is possible time period.