CS5570
Architecture of Database Management Systems

Vijay Kumar
Computer Science Electrical Engineering
University of Missouri-Kansas City
Kansas City, MO, USA.

Query Processing and Optimization

Query Processing: This activity involves: parsing (decomposing), validating, optimizing and executing the query. The optimizing step is optional; however, it cannot be separated from the processing activities. A query always goes through this step and optionally the system may not spend any time in optimizing the query. This may happen when the optimizing may not improve any aspect of the processing.

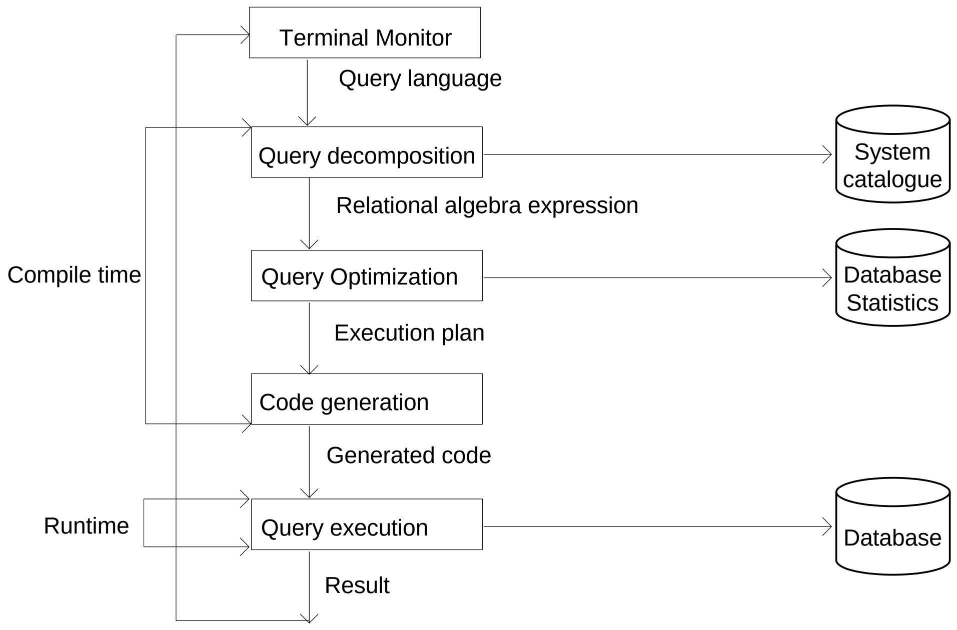


Figure 1. Query processing steps.

Query decomposition

It is the first phase of query processing. It transforms a high-level (e.g. SQL) query into a canonical relational algebra expression. It checks the syntax (format, key words, correct use of key words, etc.) and semantics (grammar). The typical stages of query decomposition are *analysis*, *normalization*, *semantic analysis*, *simplification*, and *query restructuring*.

Analysis: (a) Analyze query lexically and syntactically, (b) Verify relations and their attributes (through system catalogue), and (c) verify operation-operand compatibility, i.e., right operation on right component. Consider the following example:

Select Customer name
From Customer, Invoice
Where region = ‘Kansas City and Amount > 1000

This query could be rejected if (a) in Select Customer name is not defined and (b) any part of the predicate “*region = ‘Kansas City and Amount > 1000*” is not defined.

At the end of this step, the high-level query is transformed into some internal representation (machine dependent) that is more suitable for processing. The internal form can be a query tree (discussed later).

Normalization

This operation converts the query into a normalized form for easy manipulation. The predicate is converted into a set of "*conjunctive*" and/or *disjunctive*" forms. This conversion takes into consideration the priority of "*and*" "*or*" operators for correct conversion.

Semantic analysis

This phase rejects normalized queries that are incorrectly formulated or contradictory. A query is incorrectly formulated if the components do not contribute to the generation of the result, which may happen if some join specifications are missing. A query is contradictory if its predicate cannot be satisfied by any tuple. For example, "***region = 'Kansas City $\wedge$ region = Wichita***" is contradictory.

Correctness check algorithms

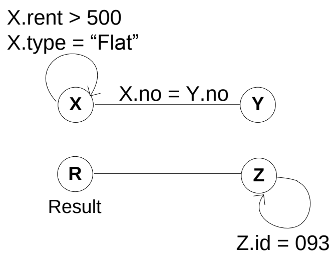
Relation connection graph construction

The query is incorrectly formulated if the graph is not connected. To construct a relation connection graph: (a) create a node for each relation, (a) a node for the result, (c) create an edge between these nodes that represents a join and edge involving one node that represents the source of projection (**Π)**.

Example

SQL query: Select A, B
From X, Y, Z
Where X.no = Y.no AND X.rent ≥ 500 AND X.type = "Flat" AND Z.id = 093;

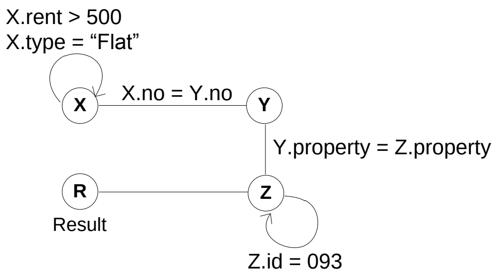
Relation connection graph



Query is incorrectly formulated because the join condition (*Y.property = Z.property*) of relations Z and Y are not included in SQL. The query is corrected by including the join condition.

Example

SQL query: Select A, B
From X, Y, Z
Where X.no = Y.no AND X.rent > 500 AND Y.property = Z.property AND
X.type = "Flat" AND Z.id = 093;



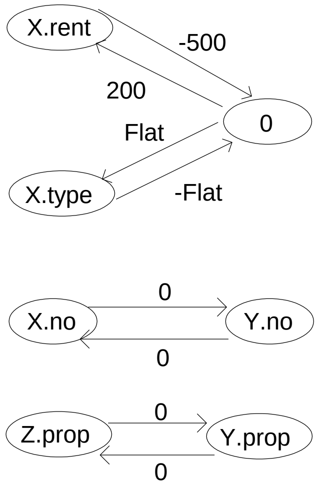
Normalized attribute connection graph construction

If the graph has a cycle for which the valuation sum is negative then the query is contradictory. Graph construction: (a) create a node for each reference to an attribute in the predicate or a constant 0, (b) connect nodes with a directed edge to represent a join, (c) connect an attribute node and a constant to represent a Selection (**σ**), (d) assign weight ***w*** to an edge a → b using the value associated with an attribute in the predicate, if it represents the inequality condition (a ≤ b + c), and weight the edge 0 a with the value –***w***, if it represents the inequality condition (a ≥ w).

Example

SQL query: Select A, B
From X, Y, Z
Where X.rent > 500 AND X.no = Y.no AND Y.property = Z.property AND
X.type = "Flat" AND c.rent < 200;

The normalized attribute connection graph for this SQL query is given below. It has a cycle between X.rent and 0 with a negative valuation sum which indicates the query is contradictory. Clearly, we cannot have a client with a rent that is both > 500 and less than < 200.



Simplification

This step (a) detects redundant qualifications, (b) eliminates common sub-expressions, and (c) transforms the query to a semantically equivalent but more easily and efficiently computable form. Typically, access restrictions, view definitions, and integrity constraints are considered at this stage, some of which may also introduce redundancy. If the user does not have the appropriate access to all the components of the query, the query is rejected. Assuming that the user has the appropriate access privileges, an initial optimization is to apply the well-known idempotency rules of Boolean algebra, such as:

$p \wedge (p) \equiv p$ $p \vee (p) \equiv p$ $p \wedge \text{false} \equiv \text{false}$ $p \vee \text{false} \equiv p$ $p \wedge \text{true} \equiv p$ $p \vee \text{true} \equiv p$

$p \wedge (\sim p) \equiv \text{false}$ $p \vee (\sim p) \equiv \text{true}$ $p \wedge (p \vee q) \equiv p$ $p \vee (p \wedge q) \equiv p$

Example:

View: Create View Staff3 as
Select Staff_no, Fname, Iname, Salary, Branch_no
From Staff
Where Branch_no = “B003”;

Query: Select *
From Staff3
Where (Branch_no = “B003” AND Salary > 20000);

This query is over a view Staff3, which is created from the base relation Staff. Thus, the view must be validated (view resolution). The query on Staff3 view is regenerated on the base relation Staff as:

Query: Select Staff_no, Fname, Iname, Salary, Branch_no
From Staff
Where (Branch_no = “B003” AND Salary > 20000) AND Branch_no = “B003”;

The predicate of this query can be simplified as:

(Branch_no = “B003” AND Salary > 20000).

Integrity constraint check.

Suppose the integrity constraint is: Managers have salary >20,000, which is created as:

Create Assertion Only_manager_salary_high
Check ((position <> “Manager” AND Salary < 20000)
OR ((position <> “Manager” AND Salary > 20000));

The effect of this on the query:

Select Staff_no, Fname, Iname, Salary, Branch_no
From Staff
Where position = “Manager” AND Salary < 15000;

This query violates the constraint.

Query restructuring

A query can be restructured by rearranging the order of relational algebra operations in the query. This reordering is achieved by the following transformation rules:

Laws involving $\bowtie$ and $\times$

1. **Cascade of σ**: $\sigma_{(c1 \text{ and } c2)}(r) \equiv \sigma_{c1}(\sigma_{c2}(r))$.
2. **Commutativity of σ**: $\sigma_{c1}(\sigma_{c2}(r)) \equiv \sigma_{c2}(\sigma_{c1}(r))$.
3. **Cascade of Π**: $\Pi_{A1}(\Pi_{A2}(\ldots(\Pi_{An}(r))\ldots)) \equiv \Pi_{A1}(r)$.
4. **Commuting Π and σ**: $\Pi_{A1, A2, \ldots, An}(\sigma_c(R)) \equiv \sigma_c(\Pi_{A1, A2, \ldots, An}(r))$.
5. **Commutative laws for $\bowtie$ and ×**: Join condition **c**. and attribute sets *A* and *B*, then
$A \bowtie_c B \equiv B \bowtie_c A$. $A \bowtie B \equiv B \bowtie A$ and $A \times B \equiv B \times A$.
6. **Commuting σ with × or $\bowtie$**: $\sigma_c(r \times s) \equiv \sigma_c(r) \times s$.
7. **Commuting Π and $\bowtie$ (or ×)**: $\Pi_A(r \bowtie s) \equiv \Pi_{Ai}(r) \bowtie \Pi_{Aj}(s)$.

8. **Commutativity of set operations**: The set operations $\cup$ and $\cap$ are commutative but – is not.
9. Associative laws for $\bowtie$, $\times$, $\cup$, and $\cap$: If *A*, *B* and *C* are sets of attributes and **c1** and **c2** are conditions then $(A \bowtie_{c1} B) \bowtie_{c2} C \equiv A \bowtie_{c1}(B \bowtie_{c2} C)$, $(A \times B) \times C \equiv A \times (B \times C)$, and similarly for the other two operators.
10. **Commuting σ with set operations**: $\sigma_C(r \Theta s) \equiv \sigma_C(r) \Theta \sigma_C(s)$.
11. **Commuting Π with $\cup$**: $\Pi_A (r \cup s) \equiv \Pi_A (r) \cup \Pi_A (s)$
12. **Converting a (σ, ×) sequence into $\bowtie$**: $\sigma_C(r \times s) \equiv (r \bowtie s)$

Query optimization: There are many ways (access paths) for accessing desired file/record. The optimizer tries to select the most efficient (cheapest) access path for accessing the data.

Most efficient processing: Least amount of I/O and CPU resources.

Selection of the best method: In a non-procedural language the system does the optimization at the time of execution. On the other hand in a procedural language, programmers have some flexibility in selecting the best method. For optimizing the execution of a query the programmer must know:

- file organization
- record access mechanism and primary or secondary key.
- data location on disk.
- data access limitations.

Consider relations *r(AB)* and *s(CD)*. We require $r \bowtie s$.

Method 1

a. Load next record of *r* in RAM.
b. Load all records of *s*, one at a time and concatenate with *r*.
c. All records of *r* concatenated?
   NO: goto a.
   YES: exit (the result in RAM or on disk).

Performance: Too many accesses.

Method 2: Improvement

a. Load as many blocks of *r* as possible leaving room for one block of *s*.
b. Run through the *s* file completely one block at a time.

Performance: Reduces the number of times s blocks are loaded by a factor of equal to the number of r records than can fit in main memory.

Quantification

Let n_r be the cardinality of *r* and n_s be the cardinality of *s*.

Then

No. of records of *r* than can fit in a block = b_r.

No. of records of s than can fit in a block = b_s.

No. of blocks main memory can hold = m.
Total no. of block accesses necessary to read $r = n_r/b_r$.

Suppose 1 memory block is used for loading blocks from s and m-1 memory blocks are available to r. In performing $r \bowtie s$, s file will be read $\frac{n_r}{b_r(m-1)}$ and each time it will require n_s/b_S block accesses. So we have two parameters (a) *the number of times a file block is loaded in the memory blocks* and (b) *the number of accesses for a file block*. Thus, in loading blocks from s file, n_s/b_S blocks will be accessed (loaded) and the total number of times these blocks will be loaded in memory blocks. The total number of block accesses is *then* $n_r/b_r + n_r/b_r \times n_S/(b_S (m-1))$, that is $\frac{n_r}{b_r}(1+\frac{n_s}{b_s(m-1)})$. If $n_r = n_S = 10{,}000$, $b_r = b_S = 5$ and $m = 100$, the number of accesses required to compute the product = 42,000. Transfer rate = 20 blocks/sec., then it will take 35 minutes. In reality m will be very large so the time will be much less than 35 mins., but the term is dominated by n_r/b_r and so the conclusion - use the file that has a larger number of blocks in the outer loop.

This example suggests that several points must be considered in the implementation of some of the relational algebra operations. The most expensive operation is $\bowtie$ then $\times$ and then σ. We will consider their implementation with respect to optimization. We discuss the first approach where the system is responsible for selecting the best processing strategy.

Implementation of Join: Join involves two relations. We analyze the nature of operations for implementing a join. We will refer to the relations *r(AB)* and *s(CD).*

If a query requires that a $\times$ is to be printed then this involves accessing both relations many times as explained earlier. This may be optimized by selecting the smaller relation to be used in the inner loop if both relations cannot fit in memory. We look at a common query:

Query: Find A where B = C and D = 99.

This can be expressed as: $\Pi_A(\sigma_{B = C \text{ and } D = 99} (r \times s))$. Since $D = 99$ is meaningful for s, we migrate $D = 99$ inside $(r \times s)$: $\Pi_A(\sigma_{B = C} (r \times \sigma_{D = 99} (s)))$.

The second expression is a significant improvement so far as data transfer is concerned. Again, since there is a σ and a $\times$, this can be treated as **equijoin** with join condition $B = C$. If we do that then our expression may be written as:

$$\Pi_A(r \bowtie_{B = C} (\sigma_{D = 99} (s))).$$

This expression can be interpreted as follows: It asks for A's that are associated with $D = 99$. The connecting path between A's and D's is defined by $B = C$. The above expression cannot be simplified any further. So we try to implement this expression.

Method 1: If there is no index defined on any of the attributes *(ABCD)* then the $\bowtie$ can be done first using a sequential search. This will require 2,000 blocks accesses (recall n_S/b_S). From this set we only need C since D is not used in any subsequent condition/operation. This can be done in the memory. The set of C values become set of B value so we can scan r using C values and this will also require 2,000 block accesses (recall n_r/b_r). So the re-written query processing will require 4,000 accesses which is much less than 42,000 accesses.

Method 2: If there are indices on all attributes then right blocks are selected and transferred to RAM. This will further reduce the total number of accesses (< 4,000).

Method 3: If *r* and *s* are large and neither is small enough to fit in RAM. In this case we can sort *r* on *B* values and *s* on *C* values. We can then run through each file once comparing equality. This will require a total of *(nr/br + ns/bs)* accesses. But the sorting time that extends over *m* (RAM) blocks is proportional to **m log m**, which dominates the time to compare the sorted files.

There may be, depending upon the resource capacity, several other ways of optimizing the processing time of this type of queries.

Heuristic Optimization: In this method relational algebra expressions are expressed in equivalent expressions that take much less time and resource to process.

Application of these rules does not guarantee an optimal expression, but it does reduce the processing overheads. As mentioned before, the general idea is to reduce the size of the operands of a binary operator (union, product, intersection etc.) as much as possible. Some special cases occur when the binary operation has operands that are σs and/or Πs applied to leaves of the tree. We must consider carefully how the binary operation is to be done, and in some cases we wish to incorporate the σ and Π with the binary operation. For example, if the binary operation is ∪, we can incorporate σ and Π below it in the tree with no loss of efficiency, as we must copy the operands anyway to form the ∪. However, if the binary operation is ×, with no following σ to make it an equijoin, we would prefer to do σ and Π first, leaving the result in a temporary file, as the size of the operand files greatly influence the time it takes to execute a full ×. We can present this description in the form of an algorithm:

Algorithm: OPTIMIZATION   Input: A relational algebra expression tree.
Output: A program for evaluating that expression.

Steps:

1. Separate each σ, $\sigma_{c1 \text{ and } c2 \ldots cn}(A)$ into the cascade $\sigma_{c1}(\sigma_{c2}(\ldots\sigma_{cn}(A)\ldots))$.
2. For each σ move the selection as far down the tree as possible.
3. For each Π move the Π as far down as possible. Also, eliminate a Π if it projects an expression onto all its attributes.
4. Combine cascades of σs & Πs into a single σ, a single Π, or a σ followed by a Π.
5. Partition the interior nodes of the resulting trees into groups, as follows. Every interior node representing a binary operator ×, ∪, or - is in a group along with any of its immediate ancestors that are labeled by a unary operator (σ or Π). Also include in the group any chain of descendant labeled by unary operators and terminating at a leaf, except in the case that the binary operator is a × and not followed by a σ that combines with the × to form an equijoin.
6. Produce a program consisting of a step to evaluate each group in any order such that no group is evaluated prior to its descendant groups.

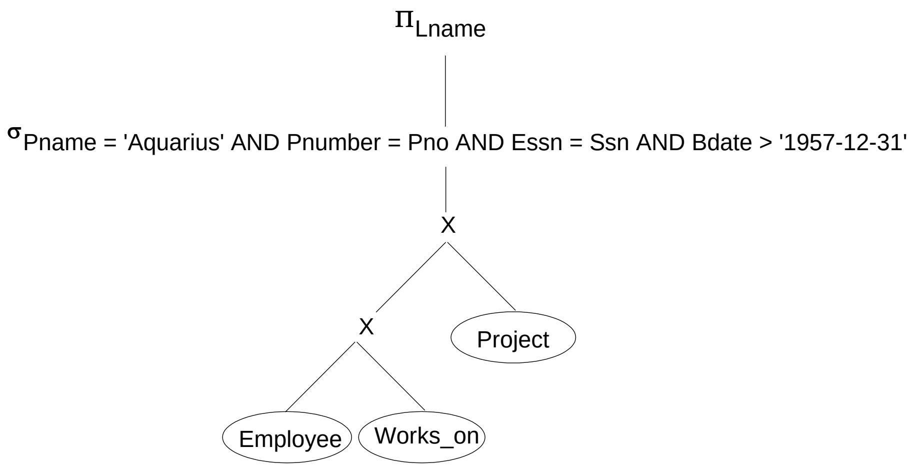
Example 1:

Select Lname
From Employee, Works_on, Project
Where Pname = “Aquarius” AND Pnumber = Pno AND Essn = Ssn AND Bdate > “1957-12-31”;

Cannonical relational algebra expression

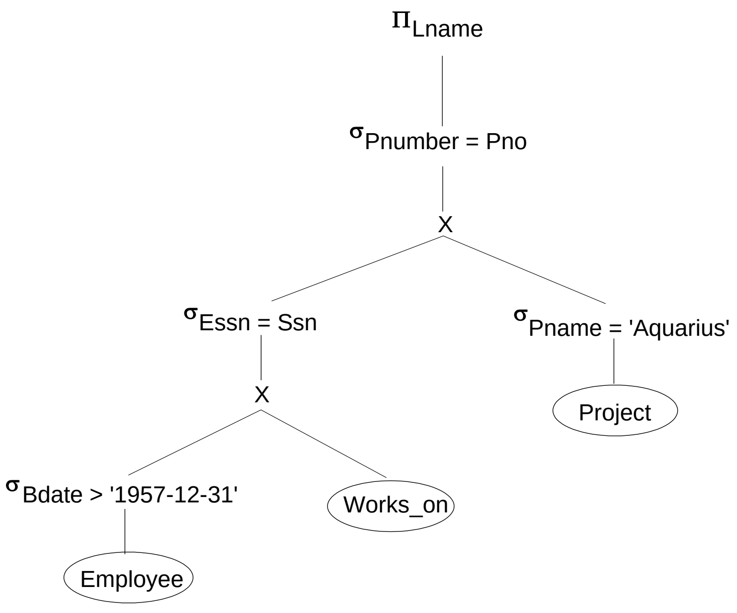
$\Pi_{\text{Lname}}(\sigma_{\text{Pname = 'Aquarius' AND Pnumber = Pno AND Essn = Ssn AND Bdate > '1957-12-31'}})$

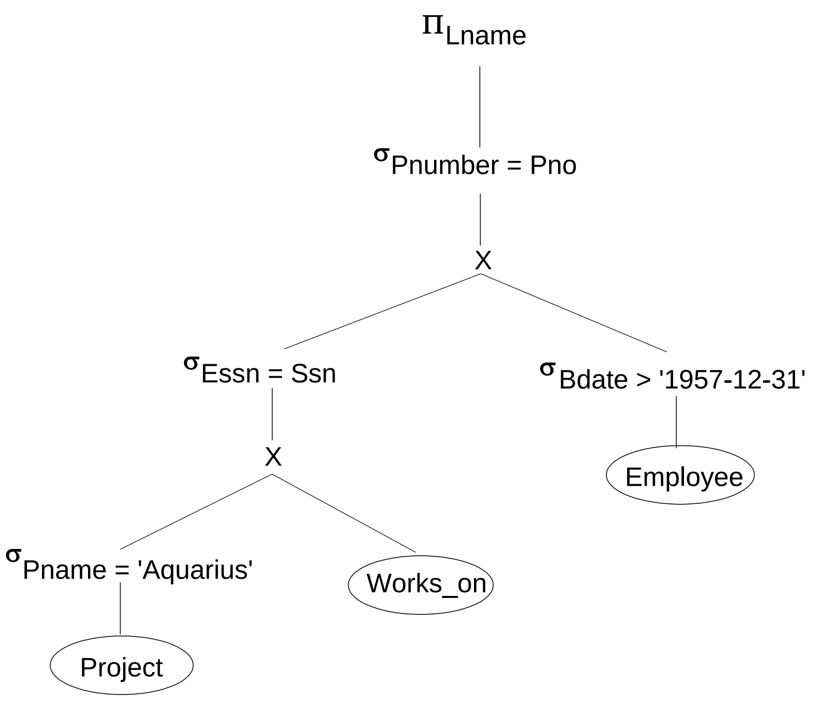
Query tree for this canonical expression



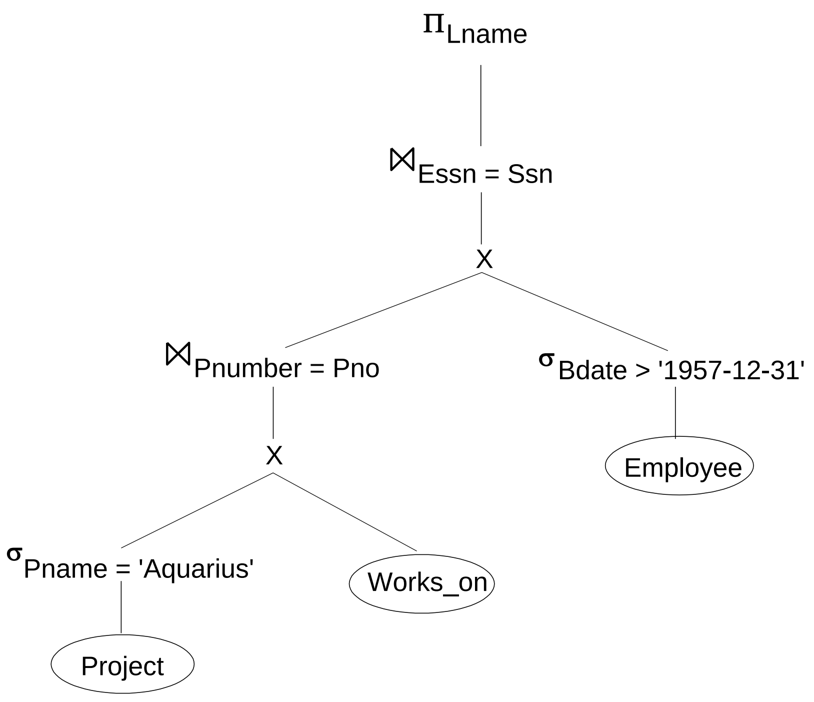
πLname
σPname = 'Aquarius' AND Pnumber = Pno AND Essn = Ssn AND Bdate > '1957-12-31'
X
X
Project
Employee
Works_on

The optimization begins by reorganizing relational algebra operations in the tree. The reorganization is achieved with the help of transformation rules. One of the first tasks is to bring all monadic operations (Π and σ) to the leaf level and then convert X to Join. The next series of trees represent successive reorganization of these operations leading to the final optimized tree.

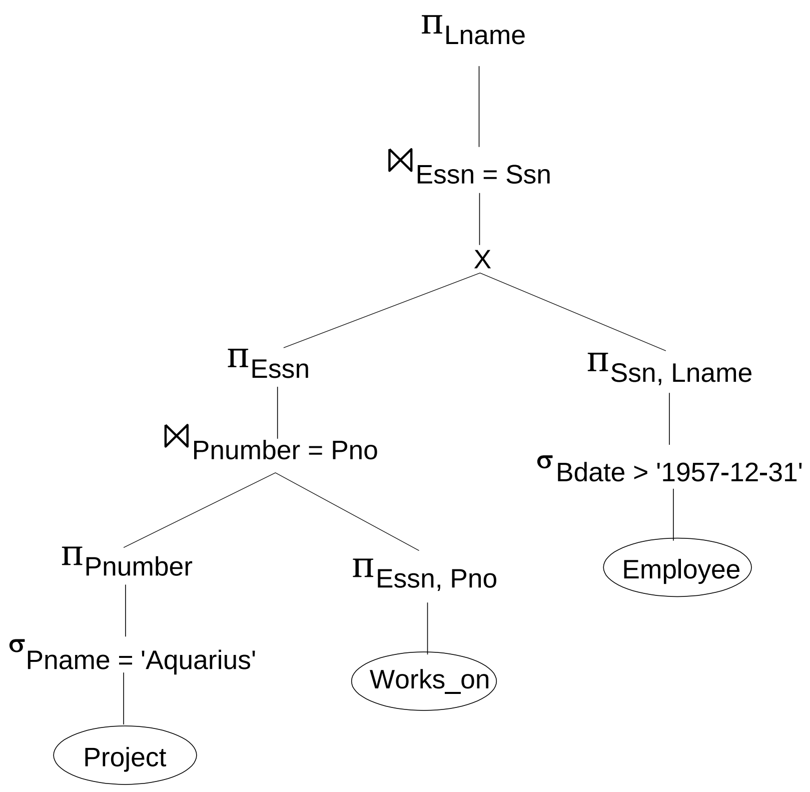




We remove cross products (×) with joins.



Finally, the necessary attributes are projected out from these relations.



Example 2: **Schema**

BOOK (TITLE, AUTHOR, PNAME, LIB_NO)
PUBLISHERS (PNAME, PADDR, PCITY)
BORROWERS (NAME, ADDR, CITY, CARD_NO)
LOANS (CARD_NO, LIB_NO, DATE)

Query: **Get the name, address, city, card_no and borrowing data of the borrower and title, author, publisher's name and library of congress no. (LIB_NO), and then list the title of books borrowed before 1/1/82.**

The relational algebra expression:

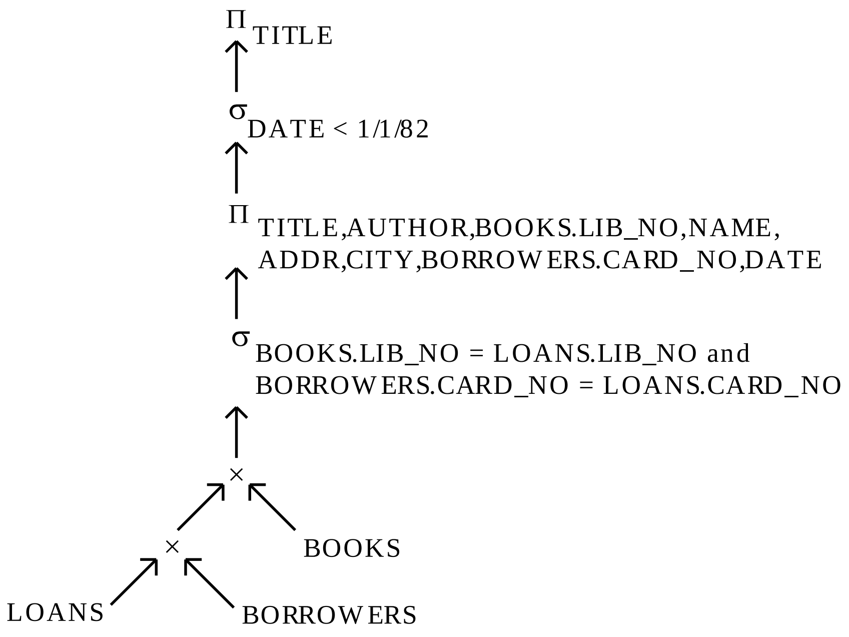
BLOAN = $\Pi_S (\sigma_C (\text{LOANS} \bowtie \text{BORROWER} \bowtie \text{BOOKS}))$

BTITLE = $\Pi_{TITLE} (\sigma_{date < 1/1/82} (\text{BLOAN}))$

Where

s = TITLE, AUTHOR, PNAME, LIB_NO, NAME, ADDR, CITY, CARD_NO, DATE
c = BORROWERS.CARD_NO = LOANS.CARD_NO and
BOOKS.LIB_NO = LOANS.LIB_NO

Query tree for this expression


Π TITLE
σ DATE < 1/1/82
Π TITLE,AUTHOR,BOOKS.LIB_NO,NAME, ADDR,CITY,BORROWERS.CARD_NO,DATE
σ BOOKS.LIB_NO = LOANS.LIB_NO and BORROWERS.CARD_NO = LOANS.CARD_NO
×
×
BOOKS
LOANS
BORROWERS

Optimization: Split c into two individual conditions:

BORROWERS.CARD_NO = LOANS.CARD_NO
BOOKS.LIB_NO = LOANS.LIB_NO

There are three σs so move each of them as far as down the tree as possible. $\sigma_{\text{DATE} < 1/1/82}$ moves below Π and the two σs by rules 4 & 5. This σ then applies to (LOANS × BORROWERS) × BOOKS. Since the DATE is the only attribute mentioned by the σ, and date is an attribute of LOANS, we can replace

$\sigma_{\text{DATE} < 1/1/82}$ ((LOANS × BORROWER) × BOOKS) by

($\sigma_{\text{DATE} < 1/1/82}$ (LOANS × BORROWER)) × BOOKS and then by

(($\sigma_{\text{DATE} < 1/1/82}$ (LOANS)) × BORROWER) × BOOKS

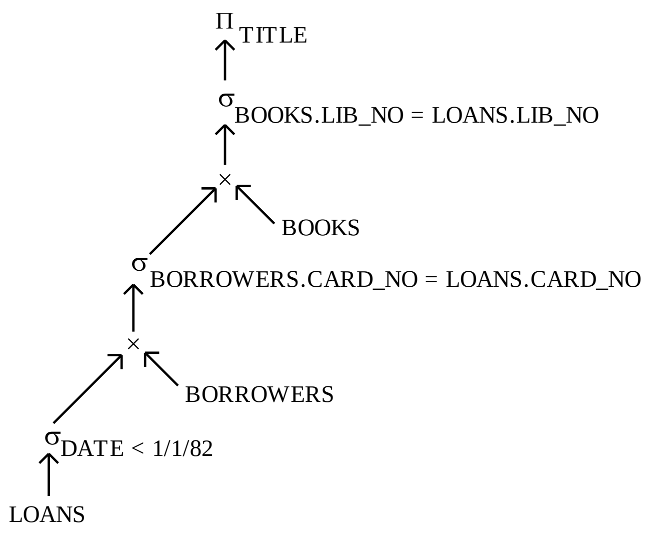
We have now moved this σ as far down as possible. The σ with condition BOOKS.LIB_NO = LOANS.LIB_NO cannot be moved below either ×, since it involves an attribute of BOOKS and an attribute not belonging to BOOKS. However, the selection on

BORROWERS.CARD_NO = LOANS.CARD_NO

can be moved down to apply to $\sigma_{\text{date} < 1/1/82}$ (LOANS) × BORROWER

Note that LOANS.CARD_NO is the name of an attribute of $\sigma_{\text{date} < 1/1/82}$ (LOANS) since it is an attribute of LOANS, and the result of a ⋈ takes its attributes to be the same as those of the expression to which the ⋈ is applied.

Next, we can combine the two Πs into one, Π_{TITLE}, by rule 3. The resulting tree is shown below.


Π TITLE
σ BOOKS.LIB_NO = LOANS.LIB_NO
$\times$
BOOKS
σ BORROWERS.CARD_NO = LOANS.CARD_NO
$\times$
BORROWERS
σ DATE < 1/1/82
LOANS

Then by extending rule 5 we can replace and by the cascade

Π_{TITLE} ($\sigma_{\text{BOOKS.LIB\_NO = LOANS.LIB\_NO}}$)

$\Pi_{\text{TITLE, BOOKS.LIB\_NO, LOANS.LIB\_NO}}$

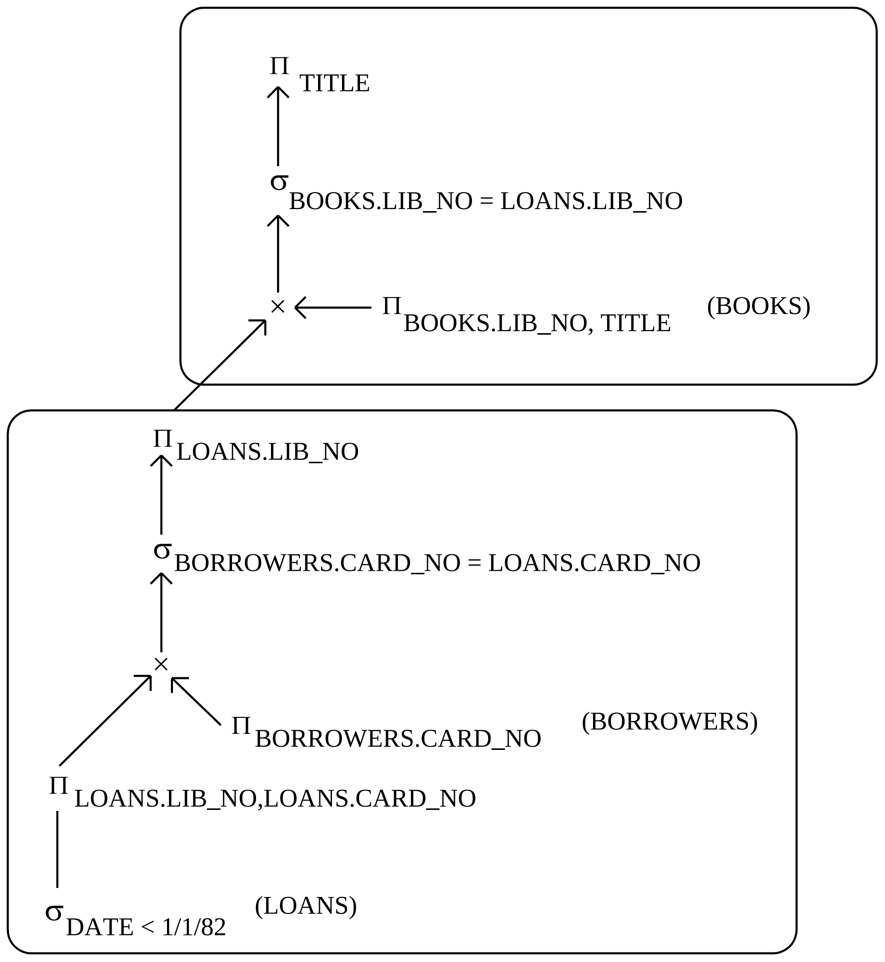
We apply rule 9 to replace the last of these Π_S by $\Pi_{\text{TITLE, BOOKS.LIB\_NO}}$ applied to BOOKS, and $\Pi_{\text{LOANS.LIB\_NO}}$ applied to the left operand of the higher × in the above figure.

The latter Π interacts with the σ below it by the extended rule 5 to produce the cascade

$\Pi_{\text{LOANS.LIB\_NO}}$ $\sigma_{\text{BORROWERS.CARD\_NO = LOANS.CARD\_NO}}$

$\Pi_{\text{LOANS.LIB\_NO, BORROWERS.CARD\_NO, LOANS.CARD\_NO}}$

The last of these Πs passes through the × by rule 9 and passes partially through the σ by the extended rule 5. We then discover that in $\Pi_{\text{LOANS.LIB\_NO, LOANS.CARD\_NO, DATE}}$ the Π is superfluous, since all attributes of LOANS are mentioned. We therefore eliminate this Π. The final tree is shown below.


Π TITLE
σ BOOKS.LIB_NO = LOANS.LIB_NO
×
Π BOOKS.LIB_NO, TITLE
(BOOKS)
Π LOANS.LIB_NO
σ BORROWERS.CARD_NO = LOANS.CARD_NO
×
Π BORROWERS.CARD_NO
(BORROWERS)
Π LOANS.LIB_NO,LOANS.CARD_NO
σ DATE < 1/1/82
(LOANS)