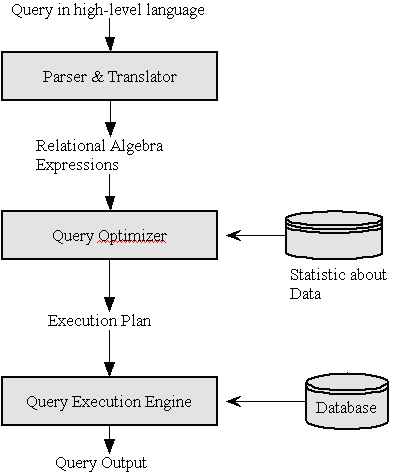
**Query Processing and Optimization**

Query processing is a set of activities involving in getting the result of a query expressed in a high-level language. These activities includes parsing the queries and translate them into expressions that can be implemented at the physical level of the file system,

**1. Overview of Query Processing**

The main steps in processing a high-level query are illustrated in figure 1



Steps in query processing process

* **Parser and Translator**

The functions of Query Parser is parsing and translating a given high-level language query into its immediate form such as relational algebra expressions. The parser need to check for the syntax of the query and also check for the semantic of the query ( it means verifying the relation names, the attribute names in the query are the names of relations and attributes in the database). A parse-tree of the query is constructed and then translated into **relational algebra expression**.

* **Query Optimizer**

A relational algebra expression of a query specifies only partially how to evaluate a query, there are several ways to evaluate an relational algebra expression.

For example, consider the query :

**SELECT Salary FROM EMPLOYEE WHERE Salary >= 50000 ;**

The possible relational algebra expressions for this query are:

* ΠSalary(σSalary>=50000(EMPLOYEE))
* σSalary>=50000(ΠSalaryEMPLOYEE)

Further, each relational algebra operation can be executed using various algorithms. For example, to implement the preceding selection, we can do a linear search in the EMPLOYEE file to retrieve the tuples with Salary >= 50000. However, if an index available on the Salary attribute, we can use the index to locate the tuples. Different algorithms might have different cost.

Query Execution Plan:

In order to specify fully how to evaluate a query, the system is responsible for constructing a **query execution plan** (An annotated expression specifying detailed evaluation strategy) which is made up of the relational algebra expression and the detailed algorithms to evaluate each operation in that expression. Moreover, the selected plan should minimize the cost of query evaluation. Among all semantically equivalent expressions, the one with the least costly evaluation plan is chosen. Cost estimate of a plan is based on statistical information in the system catalogs.

The process of choosing a suitable query execution plan is known as query optimization This process is performed by Query Optimizer.

* **Query Execution Engine**

Once the query plan is chosen, the Query Execution Engine lastly take the plan, executes that plan and returns the answer of the query.

## Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

* Select
* Project
* Union
* Set different
* Cartesian product
* Rename

We will discuss all these operations in the following sections.

## Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

**Notation** − σ*p*(r)

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < ,  >,  ≤.

**For example** −

σsubject = "database"(Books)

**Output** − Selects tuples from books where subject is 'database'.

σsubject = "database" and price = "450"(Books)

**Output** − Selects tuples from books where subject is 'database' and 'price' is 450.

σsubject = "database" and price = "450" or year > "2010"(Books)

**Output** − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

## Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − ∏A1, A2, An (r)

Where A1, A2 , An are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

**For example** −

∏subject, author (Books)

Selects and projects columns named as subject and author from the relation Books.

## Union Operation (∪)

It performs binary union between two given relations and is defined as −

r ∪ s = { t | t ∈ r or t ∈ s}

**Notion**  r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

* **r**, and **s** must have the same number of attributes.
* Attribute domains must be compatible.
* Duplicate tuples are automatically eliminated.

∏ author (Books) ∪ ∏ author (Articles)

**Output** − Projects the names of the authors who have either written a book or an article or both.

## Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

**Notation**  **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

∏ author (Books) − ∏ author (Articles)

**Output** − Provides the name of authors who have written books but not articles.

## Cartesian Product (Χ)

Combines information of two different relations into one.

**Notation**  r Χ s

Where **r** and **s** are relations and their output will be defined as −

r Χ s = { q t | q ∈ r and t ∈ s}

σauthor = 'tutorialspoint'(Books Χ Articles)

**Output** − Yields a relation, which shows all the books and articles written by tutorialspoint.

## Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

**Notation** − *ρ* x (E)

Where the result of expression **E** is saved with name of **x**.

**The set intersection operation: -**finds tuples in both the relations.

* It is denoted as **∩**.

Example:  
Borrower (customer-name, loan-number)  
Depositor (customer-name, account-number)  
Customer (customer-name, street-number, customer-city)  
  
List all the customers who have both a loan and an account.

Code:

**Π customer-name (Borrower) ∩ Π customer-name (Depositor)**

**The natural join operation: -**it is a binary operation and a combination of certain selections and a Cartesian product into one operation.

* It is denoted as |X| .
* It is associative.

It forms a Cartesian product of its two arguments.  
Then performs a selection forcing equality on those attributes those appear in both the relations.  
And finally removes duplicates attributes.  
  
r(R): r is a relation with attributes R.  
s(S): s is a relation with attributes S.  
  
If R **∩**S = Ф i.e. they have no attributes in common then **r |X| s = r X s**

**Query Optimization**

### Heuristic Query Optimization

Heuristic optimization applies the rules to the initial query expression and produces the heuristically transformed query expressions.

* A query can be represented as a tree data structure. Operations are at the interior nodes and data items (tables, columns) are at the leaves.
* The query is evaluated in a depth-first pattern.
* Consider this query from Elmasri/Navathe:

**Q: For every project located in ‘Stafford’, retrieve the project number,the controlling department number, and the department manager’s last name**

SELECT PNUMBER, DNUM, LNAME

FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE DNUM=DNUMBER and MGREMPID=EMPID and

PLOCATION = 'Stafford';

Or, in relational algebra:  
http://holowczak.com/wp-content/uploads/relationalalgebraq.gif

on the following schema:

**EMPLOYEE TABLE**

**FNAME MI LNAME EMPID BDATE ADDRESS S SALARY SUPERMPID DNO**

**-------- -- ------- --------- --------- ------------------------- - ------ --------- --**

**JOHN B SMITH 123456789 09-JAN-55 731 FONDREN, HOUSTON, TX M 30000 333445555 5**

**FRANKLIN T WONG 333445555 08-DEC-45 638 VOSS,HOUSTON TX M 40000 888665555 5**

**ALICIA J ZELAYA 999887777 19-JUL-58 3321 CASTLE, SPRING, TX F 25000 987654321 4**

**JENNIFER S WALLACE 987654321 20-JUN-31 291 BERRY, BELLAIRE, TX F 43000 888665555 4**

**RAMESH K NARAYAN 666884444 15-SEP-52 975 FIRE OAK, HUMBLE, TX M 38000 333445555 5**

**JOYCE A ENGLISH 453453453 31-JUL-62 5631 RICE, HOUSTON, TX F 25000 333445555 5**

**AHMAD V JABBAR 987987987 29-MAR-59 980 DALLAS, HOUSTON, TX M 25000 987654321 4**

**JAMES E BORG 888665555 10-NOV-27 450 STONE, HOUSTON, TX M 55000 1**

**DEPARTMENT TABLE:**

**DNAME DNUMBER MGREMPID MGRSTARTD**

**--------------- --------- --------- ---------**

**HEADQUARTERS 1 888665555 19-JUN-71**

**ADMINISTRATION 4 987654321 01-JAN-85**

**RESEARCH 5 333445555 22-MAY-78**

**PROJECT TABLE**

**PNAME PNUMBER PLOCATION DNUM**

**---------------- ------- ---------- ----**

**ProductX 1 Bellaire 5**

**ProductY 2 Sugarland 5**

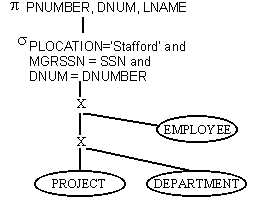
**ProductZ 3 Houston 5**

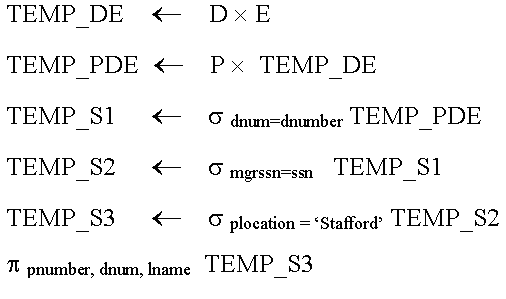
**Computerizatn. 10 Stafford 4**

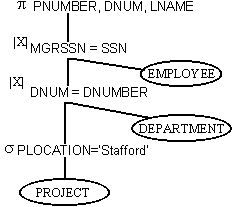
**Reorganization 20 Houston 1**

**NewBenefits 30 Stafford 4**

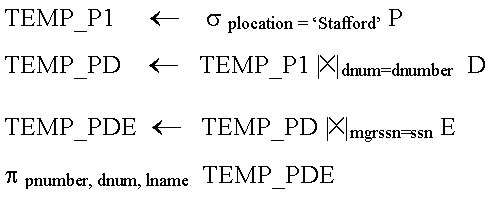
* **Which of the following query trees is more efficient ?**

  
  
This tree is evaluated in steps as follows:



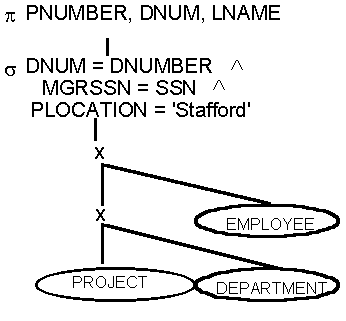
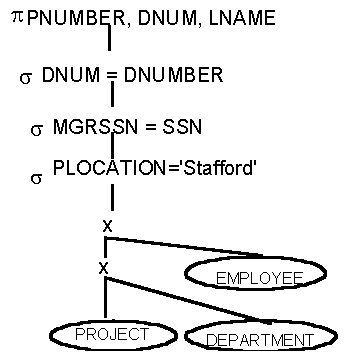
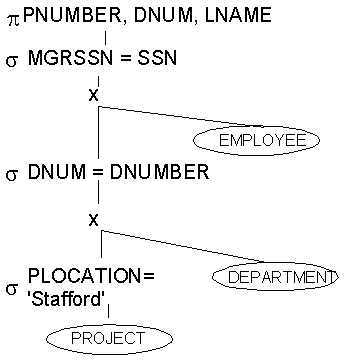
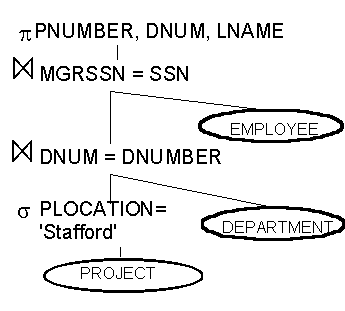


This tree is evaluated in steps as follows:



* Note the two cross product operations. These require lots of space and time (nested loops) to build.
* After the two cross products, we have a temporary table with 144 records (6 projects \* 3 departments \* 8 employees).
* An overall rule for heuristic query optimization is to perform as many select and project operations as possible before doing any joins.

Example of Heuristic Query Optimization

1. Original Query Tree  
  
  
2. Use Rule 1 to Break up Cascading Selections  
  
  
3. Commute Selection with Cross Product  
  
  
4. Combine Cross Product and Selection to form Joins  


### 2. Cost Estimates in Query Optimization

Typically, a query optimizer is not only depended on heuristic rules but also on estimating and compare the cost of executing different plans then choose the query execution plans with lowest cost.

## Measure of Query Cost

The cost of a query execution plan includes the following components:

* Access cost to secondary storage: This is the cost of searching for, reading, writing data blocks of secondary storage such as disk.
* Computation cost: This is the cost of performing in-memory operation on the data buffer during execution. This can be considered as CPU time to execute a query
* Storage cost: This is the cost of storing immediate files that are generated during execution
* Communication cost: This is the cost of transfering the query and its result from site to site ( in a distributed or parallel database system)
* Memory usage cost: Number of buffers needed during execution.

In a large database, access cost is usually the most important case since disk accesses are slow compared to in-memory operations.

In a small database, when almost data reside in the memory, the emphasis is on computation cost. In the distributed system, communication cost should be minimized.

It is difficult to include all the cost components in a cost function. Therefore, some cost functions consider only disk access cost as the reasonable measure of the cost of a query-evaluation plan.

## Catalog Information for Cost Estimation

Query optimizers use the statistic information stored in DBMS catalog to estimate the cost of a plan. The relevant catalog information about the relation includes:

* Number of tuples in a relation r; denote by nr
* Number of blocks containing tuple of relation r: br
* Size of the tuple in a relation r ( assume records in a file are all of same types): s­r
* Blocking factor of relation r which is the number of tuples that fit into one block: fr
* V(A,r) is the number of distinct value of an attribute A in a relation r. This value is the same as size of πA(r). If A is a key attribute then V(A,r) = nr
* SC(A,r) is the selection cardinality of attribute A of relation r. This is the average number of records that satisfy an equality condition on attribute A.

In addition to relation information, some information about indices is also used:

* Number of levels in index i.
* Number of lowest –level index blocks in index i ( number of blocks in leaf level of the index)

The statistical information listed here is simplified. The optimizer on real database management system might have further information to improve the accuracy of their cost estimates.

With the statistical information maintained in DBMS catalog and the measures of query cost based on number of disk accesses, we can estimate the cost for different relational algebra operations.

Now, consider a selection in EMPLOYEE file

σDeptId=1(EMPLOYEE)

The file EMPLOYEE has the following statistical information:

* f = 20 (there are 20 tuples can fit in one block)
* V(DeptID, EMPLOYEE) = 10 (there are 10 different departments)
* n = 1000 ( there are 1000 tuples in the file)

Cost for doing linear search is b = 1000/20 = 50 block accesses

Cost for doing binary search on ordering attribute DeptID:

* Average number of records that satisfy the condition is : 1000/10 = 100 records [out of 10 =1 department so out of 1000 = how many departments?]
* Number of blocks contains these tuples is: 100/20 = 5
* A binary search for the first tuple would take log250 = 6
* Thus the total cost is : 5 + 6 – 1 = 10 block accesses

**Example 2:** If the Student relation has blocks of 4K bytes, and there are 10,000 student records each 200 bytes long, then 20 records fit per block (4096/200) [200 bytes = 1 record. Then 4KB = How many records?]. We need 10000/20 or 500 blocks to hold this file in packed form

**Example 3:** Relation R is having 20,000 tuples Where R(a,b,c) Each tuple is of 190 bytes i.e. header =24 bytes, a= 8 bytes, b= 8 bytes and c=150 bytes Each block =1024 bytes , header = 24 bytes We can fit 5 tuples in 1 block (1024/190) so in 1 block = 5\*190 = 950 bytes . To store 20,000 records we will require 4000 blocks (20000/5tuples per block) So if we perform projection with eliminating C attribute then 190-150 = 40 bytes so 25 tuples can be stored in one block and 800 blocks will be required. So blocks have been reduced by factor of 1/5

http://cnx.org/contents/231e6ff4-de39-464a-bafd-0eb69b8477dd@1/Query-Processing-and-Optimizat