Ab Initio Training

Goal: -

Our Goal is to learn Ab Initio to the best of our knowledge.
Attend the client interview with confidence.
Demonstrate your exceptional Ab Initio Skills at the client place to build long-term relationship with the client.
Develop a greater carrier for your self and enjoy working in Data Warehousing and Ab Initio Parallel processing tool and feel great.

Topics to Cover In Training

Introduction
Basic Concepts: -
- **What is Ab Initio Mean?**
- **Parallel Computer Architecture.**
- **Ab Initio Product Architecture**
  - **Co>Os**
  - **GDE**
- **The Graph**
  - **Datasets**
  - **Flow**
  - **Components**
  - **Ports**
  - **Output datasets**
  - **Database**
- **DML**
  - **Datasets**
  - **Records**
  - **Fields**
  - **Metadata**
- **Explain SIMPLE GRAPH with SELECT, SORT components**
  - **What is key?**
  - **View Data**

Explain Reformat and XFR
Transform Function Syntax
Functions:
- **Is_valid**
- **Is_bzero**
- **Is_defined**
- **String_* functions**
- **Data Cleansing**

Data Aggregation using rollup
- **Init**
- **Calc loop**
- **Result**

Practical Example 1
Practical Example 2
Practical Example 3

Priority Assignment
Join
Practical Example 1
Practical Example 2
Practical Example 3

General Discussion
Error Diagnostics
- **Limit**
- **Ramp**
- **LOG**

Replicate

Basics Of Parallelism
- **Component**
- **Pipeline**
- **Data**

Partition
Multifiles
Practical Example 1
Practical Example 2
Practical Example 3

More Partitions
- **Roundrobin**
- **Hash**
- **Function**

- Range
- Load-Level

Fan-out Flows
Fan-in Flows
Practical Example 1
Practical Example 2
Practical Example 3

Hash partitioning
Hash + sort Partitioning With Join running parallel
Various Join techniques
Practical Example 1
Practical Example 2
Practical Example 3

More components
Broadcast
Gather
Replicate
Select by expr
Rollup

Practical Example 1
Practical Example 2
Practical Example 3

Review Previous days Exercises
General Discussion
More components
- De partitioning
  - Concatenation
  - Interleave
  - Gather
- Repartitioning
  - Partition Again

Practical Example 1
Practical Example 2
Practical Example 3

- Take a Close Look at How Ab Initio Runs On UNIX
  - Phases
  - Performance Issues
  - Checkpoints
  - Normalize
  - Extracting

- Cleansing
- DW to DM concept
- Agenet
- Host

Prepare and Review Of Resumes

Practical Example 1
Practical Example 2
Practical Example 3

Ab Initio Interview Process
Interview Questions

Validation Components
- Generate Data
- Validate Data

Practical Example 1
Practical Example 2
Practical Example 3

Ab Initio With RDBMS
Unloading
Loding
Db_config
Proto_config

Practical Example 1
Practical Example 2
Practical Example 3

Co>Os Administration
AB_REPORT
SANDBOX
File management UTILS
- M_env
- M_mkfs
- M_rmfs
- M_touch
- M_ls
- M_rollback

Practical Example 1
Practical Example 2
Practical Example 3

Lookup and Lookup local functions
Ab Initio String Functions
Using Local Variables
Using Data Manipulations

Practical Example 1
Practical Example 2
Practical Example 3

Shell Scripting
Environment Variables

Ab Initio:

What is Ab Initio

Ab Initio Is is a Latin word , Meaning From the beginning.

Ab Initio software helps you build large-scale data processing applications and run them in parallel environments. Ab Initio software consists of two main programs:

· Co>Operating System, which your system administrator installs on a host UNIX or Windows NT Server, as well as on processing nodes. (The host is also referred to as the control node.)

· Graphical Development Environment (GDE), which you install on your PC (client node) and configure to communicate with the host (control node).

PICTURE

BASIC TERMS of In Ab Initio

What is Dataset ?

In Simple terms dataset is a file. A file can be main frame file or any fixed or delimited files. There are various types of datasets

FIDXED EBCDIC
FIXED ASCII
DELIMITED ASCII
SAS dataset etc..

You can also think of dataset as a table in database world.

What is Component?

Component is Ab Initio Program
There are various components like SELECT, FILTER, SORT, JOIN, MERGE, DEDUP, ROLLUP, SCAN, USER DEFINED etc…

What is Port?

A port is a connection point for the input or output to a component.

What is Flow?

A flow carries a stream of data between components in a graph. Flows connect components via ports. Ab Initio supplies four kinds of flows with different patterns: straight, fan-in, fan-out , and all-to-all . We will discuss various kinds of flows as we go through this training.

What is Graph ?

A graph is a diagram that defines the various processing stages of a task and the streams of data as they move from one stage to another. Visually, stages are represented by components and streams are represented by flows. The collection of components and flows comprise an Ab Initio graph.

What is Field ?
A filed is Equivalent to column of a table in Database world. Filed is also called variable which holds value.

What is Key ?

Key is used many places in Ab Initio Development , We use key to sort the data , Join two files , Rollup data etc..

See the Below Graph Which explained Basic Terms.

AB Initio DML's

DML is an acronym for Data Manipulation Language in Ab Initio World . It is the Ab Initio programming language you use to define record formats. DML is equivalent to DDL in traditional databases. DML tells Ab Initio how to interpret your data.

Following list are various types of DML's

Delimited.dml

```
record
  decimal('|')  cust_id;
  string('|')  last_name;
  string('|')  first_name;
  string('|')  street_addr;
  string('|')  state;
  decimal('|')  zip;
  string('|')  gender;
  decimal('\n')  income;
end
```

Example of data :

297457|Alex|Neil Steven|149 Inkwell St.|KY|40541|M|0073900
901288|Andrus|Tom|165 Eboli St.|WY|60313|M|0492500
662197|Bannon|Jeffrey C|21 Compuserve St.|CO|70307|M|0140200

denorm.dml

```
record
  decimal(5) custid;
  decimal(3) num_trans;
  record
    date("YYYYMMDD") dt;
    decimal(6.2) amount;
  end transactions[num_trans];
  string(1) newline;
end
```

Example Of Data: -

12345 219970204 5.9519970209125.05
14521 119970205 15.50
12341 0
12341 319970202 9.9019970206 12.2319970210 62.75

ebcdic.dml

```
record
  ebcdic decimal(6)  cust_id;
  ebcdic string(18)  last_name;
  ebcdic string(16)  first_name;
  ebcdic string(26)  street_addr;
  ebcdic string(2)   state;
  ebcdic decimal(5)  zip;
  ebcdic string(1)   gender;
  ebcdic decimal(7)  income;
  ebcdic string(1)   newline;
end
// Rather than using the "ebcdic" keyword in every text field,
// this example uses the "ebcdic" keyword in front of the "record"
// keyword, affecting all text fields.

ebcdic record
  decimal(6)  cust_id;
  string(18)  last_name;
  string(16)  first_name;
  string(26)  street_addr;
  string(2)   state;
  decimal(5)  zip;
  string(1)   gender;
  decimal(7)  income;
  string(1)   newline;
end
```

Example Of data

òù÷ôõ÷Á“…§@@@@@@@@@@@@@@Õ…‰“@â£…¥…
•@@@@@ñôù@É•'¦…“‘“@â£K@@@@@@@@@@@@ÒèôðõôñÔðð÷óùðð%ùðñòøøÁ•„™¤
¢@@@@@@@@@@@@@ã–”@@@@@@@@@@@@@@ñöð@Å‚–“‰@

fixed.dml

```
record
  decimal(6)  cust_id;          // Customer ID
  string(18)  last_name;        // Last name
  string(16)  first_name;       // First name
  string(26)  street_addr;      // Street address
  string(2)   state;            // State
  decimal(5)  zip;              // Zipcode
  string(1)   gender;           // Gender (M = male; F = female)
  decimal(7)  income;           // Income (in dollars)
  string(1)   newline;
end
```

Example Of data

```
297457Alex           Neil Steven     149 Inkwell St.          KY40541M0073900
901288Andrus           Tom           165 Eboli St.           WY60313M0492500
662197Bannon          Jeffrey C      21 Compuserve St.        CO70307M0140200
```

unix-text.ml

string("\n")

Example Of data

This is text as you might
find it on a computer running a

win-text.dml

string("\r\n")

Refrer to the Graph Types.mp to define dml's and View the data

FILTER BY EXPRESSION

Following Graph is our First Ab Initio Graph. This graph processes a file to produce customers whose income is greater than $5000.

INPUT DML: -

```
record
  decimal(6)  cust_id;                  // Customer ID
  string(18)  last_name;      // Last name
  string(16)  first_name;     // First name
  string(26)  street_addr;    // Street address
  string(2)   state;          // State
  decimal(5)  zip;            // Zipcode
  string(1)   gender;                   // Gender (M = male; F = female)
  decimal(7)  income;                   // Income (in dollars)
  string(1)   newline;
end
```

See the Attached Graph.

Code: -
This Graph contains no code.

Performance / Interview Question?
Don't use filter by Expression. Most of the components has embedded filter by expression called select expression use embedded select instead of Filter by expression if possible to improve performance.

Exercise: Use Filter By Express to split the above customer information into two files one with Males and Females.

Transformation With Reformat (XFR)

What is XFR?
You write your code (logic) in XFR. Your code file extension is ..xfr. All the Transform components use XFR to run.

REFORMAT COMPONENT: -

As name suggests reformat means changing the format of input file and produce the desired output file. Example if you have 10 fields in input file you want out put file with 5 out of those 10 input fields then you will use reformat component.

With reformat you can derive new fields. See the examples below.

Example 1: Customer info Dml

```
record
  decimal(6)  cust_id;              // Customer ID
  string(18)  last_name;    // Last name
  string(16)  first_name;   // First name
  string(26)  street_addr;  // Street address
  string(2)   state;            // State
  decimal(5)  zip;              // Zipcode
  string(1)   gender;               // Gender (M = male; F = female)
  decimal(7)  income;               // Income (in dollars)
  string(1)   newline;
end
```

INPUT DATA
297457|Alex|Neil Steven|149 Inkwell St.|KY|40541|M|0073900
901288|Andrus|Tom|165 Eboli St.|WY|60313|M|0492500
662197|Bannon|Jeffrey C|21 Compuserve St.|CO|70307|M|0140200

Reformat above DML like this

```
record
  decimal(6)  cust_id;              // Customer ID
  string(18)  last_name;    // Last name
  string(16)  first_name;   // First name
  string(1)   gender;               // Gender (M = male; F = female)
  decimal(7)  income;               // Income (in dollars)
  string(1)   newline;
end
```

YOUR OUTPUT DATA LOOK's LIKE this

297457|Alex|Neil Steven|M|0073900
901288|Andrus|Tom|M|0492500
662197|Bannon|Jeffrey C|M|0140200

XFR Code For Above Example: -

```
/*Reformat operation*/
out::reformat(in) =
begin
  out.cust_id :: in.cust_id;
  out.last_name :: in.last_name;
  out.first_name :: in.first_name;
  out.gender :: in.gender;
  out.income :: in.income;
  out.newline :: in.newline;
end;
```

Reformat Example With Derived Field :-

Reformat customer DML like this , We are deriving new field called full_address which is concatenation Street_address , state, zip into one line

```
record
 decimal(6)  cust_id;              // Customer ID
 string(18)  last_name;    // Last name
 string(16)  first_name;   // First name
String(33) Full_address      ---------DERIVED FILED,
 string(1)   gender;               // Gender (M = male; F = female)
 decimal(7)  income;               // Income (in dollars)
 string(1)   newline;
end
```

XFR Code For Above Example: -

```
/*Reformat operation*/
out::reformat(in) =
begin
  out.cust_id :: in.cust_id;
  out.last_name :: in.last_name;
  out.first_name :: in.first_name;
  out.Full_address :1: string_concat ( in.street_addr,in.state,in.zip);
  out.Full_address :2: "NO Address Found";
  out.gender :: in.gender;
  out.income :: in.income;
  out.newline :: in.newline;
end;
```

In the above code string_concat is Ab Initio built-in function, Read help for all built in function. They are similar to C-Programming functions. Also note Priority Assignments 1 and 2 are like Case statements in SQL, If 1 is success take it else use 2.

Caution: - AB INITIO DML names are case sensitive.

Generate Records: -

Generate Records generates a specified number of data records with fields of specified lengths and types.

You can let Generate Records generate random values within the specified length and type for each field, or you can control various aspects of the generated values using command line option of Generate Records component. Typically, the output of Generate Records is used for testing a graph.

Example: -

Input DML:-

```
record
      decimal(6)  cust_id;
      string(18)  last_name;
      string(16)  first_name;
      string(26)  street_addr;
      decimal(2)   state_code;
      decimal(5)  zip;
      string(1)   gender="M";
      decimal(7)  income;
      date("MM/DD/YYYY") dob;
      string(1)   newline="\n";
 end
```

Set num_records option to 10000

Set command Line option as follows:-

-sequential cust_id 350000 -minimum state_code 1 -maximum state_code 50 -minimum income 100 –maximum income 100000 -default gender -default newline

Above Command line telling generate records component to generate 10,000 records , generate cust_id's sequentially starting from 350,000 , set state_code between 1 to 50 and income between 100 to 100000 and keep default values for gender and newline.

Exercises for reformat: -

Generate 50000 records with above DML in /data/abwork/your_dir/in_file1.dat

use following command line and DML

-sequential cust_id 350000 -minimum state_code 1 -maximum state_code 50 -minimum income 100 –maximum income 100000 -default gender -default newline

```
record
      decimal(6)  cust_id;
      string(18)  last_name;
      string(16)  first_name;
```

```
      string(26)  street_addr;
      decimal(2)   state_code;
      decimal(5)  zip;
      string(1)   gender="M";
      decimal(7)  income;
      date("MM/DD/YYYY") dob;
      string(1)   newline="\n";
    end
```

Generate 100000 records with above DML in /data/abwork/your_dir/in_file2.dat
use following command line and DML

-sequential cust_id 350000 -minimum state_code 1 -maximum state_code 50 -minimum income 100 -maximum income 100000 -default gender -default newline

```
record
      decimal(6)  cust_id;
      string(18)  last_name;
      string(16)  first_name;
      string(26)  street_addr;
      decimal(2)   state_code;
      decimal(5)  zip;
      string(1)   gender="F";
      decimal(7)  income;
      date("MM/DD/YYYY") dob;
      string(1)   newline="\n";
    end
```

Develop Following Graphs

Exercise 1 (GRAPH 1) : -

Use Unix cat command to make above generated data into one file like this
Cd /data/abwork/your_dir
Cat in_file1.dat in_file2.dat >> in_file.dat

Use following input DML to map in_file.dat

input DML :-

```
record
      decimal(6)  cust_id;
      string(18)  last_name;
      string(16)  first_name;
      string(26)  street_addr;
      decimal(2)   state_code;
      decimal(5)  zip;
      string(1)   gender;
      decimal(7)  income;
      date("MM/DD/YYYY") dob;
      string(1)   newline="\n";
    end
```

Define Output file with following DML
Set your output file to file:/data/abwork/your_dir/reform_ex1.out

OUT PUT DML :-

```
record
      decimal(6)  cust_id;
```

```
      string(18)  last_name;
      string(16)  first_name;
      string(26)  street_addr;
      decimal(2)   state_code;
      decimal(5)  zip;
      string(1)   gender;
      decimal(7)  income;
      date("MM/DD/YYYY") dob;
      decimal(3) age;
      string(1) minor_flag;
      string(1)   newline="\n";
    end
```

1) Derive a field called Age using his dob

use following expression to get age

```
 ((date("MM/DD/YYYY"))"03/17/2003"  - (date("MM/DD/YYYY"))in0.dob);
```

2) Derive a field called minor_flag , Set this flag to "Y" if age is less than 18 or set it to "N" is
age is >= 18

Exercise 2 (GRAPH 2): -

INPUT DML (same as Above Example)

```
record
      decimal(6)  cust_id;
      string(18)  last_name;
      string(16)  first_name;
      string(26)  street_addr;
      decimal(2)   state_code;
      decimal(5)  zip;
      string(1)   gender;
      decimal(7)  income;
      date("MM/DD/YYYY") dob;
      string(1)   newline="\n";
    end
```

OUTPUT DML

```
record
      decimal(6)  cust_id;
      string(18)  last_name;
      string(16)  first_name;
      string(26)  street_addr;
      decimal(2)   state_code;
      decimal(5)  zip;
      string(1)   gender;
      decimal(7)  income;
      date("MM/DD/YYYY") dob;
      decimal(5) score;
      string(1)   newline="\n";
    end
```

Based on his gender derive a filed called score, Business logic to derive score is

```
if (in.gender == "M") score = income / 2000;
if (in.gender == "M") score = income / 2000;
if (in.gender == "F") score = income / 2000 + 500;
```

Exercise 3 (GRAPH 2): -

INPUT DML (same as Above Example)

```
record
      decimal(6)  cust_id;
      string(18)  last_name;
      string(16)  first_name;
      string(26)  street_addr;
      decimal(2)   state_code;
      decimal(5)  zip;
      string(1)   gender;
      decimal(7)  income;
      date("MM/DD/YYYY") dob;
      string(1)   newline="\n";
    end
```

OUTPUT DML :-

```
record
      decimal(6)  cust_id;
      string(18)  last_name;
      string(16)  first_name;
      string(26)  street_addr;
      decimal(2)   state_code;
      decimal(5)  zip;
      string(1)   gender;
      decimal(7)  income;
      date("MM/DD/YYYY") dob;
      decimal(2) dayb;
      decimal(2) monthb;
      decimal(2) yearb;
      string(1)   newline="\n";
    end
```

Use data functions to find day born and month born and year born of above customers into derived fields dayb, monthb, yearb respectively.

ROLLUP

What is rollup?

Rollup summarize groups of data records. It is like Group by operation in SQL. The Best Way to understand rollup is using an example.

Let us say you have input dataset with following DML

The input dataset has records of this format:

```
record
  string(" ")   cust_name;
  decimal(" ")  purchase;
  decimal(" ")  age;
  string("\n")  coupon;
end;
```

A group of records like this:

Cust_name	purchase	age	coupon
Steve	100	13	Y
Steve	200	34	N
Kathy	200	38	N
Kathy	400	70	N

We would like to rollup these records by the key field to produce Records of this format:

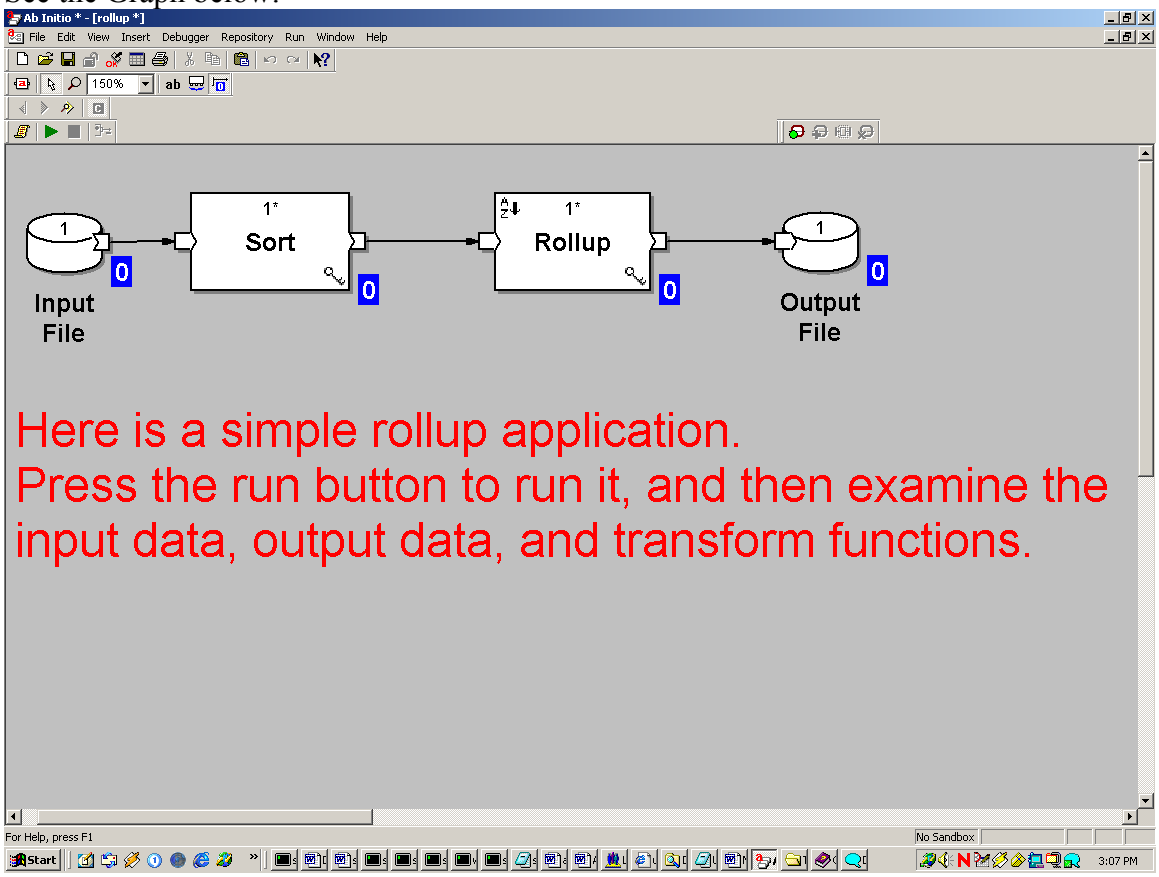
```
record
  string(" ") cust_name;
  decimal(" ") total_purchases;
  string("\n") ever_used_coupon;
end;
```

We want to see the output like this

Cust_name	total_purchases	ever_used_coupon
Steve	300	Y
Kathy	600	N

The total purchases field will contain the sum of all of the purchase field values for all records with the same key. The ever_used_coupon field will be "Y" if the customer uses a coupon in any transaction, or "N" otherwise.

See the Graph below:-



In this Graph we are using Sort Component, which is required before rollup. Sort requires a key to sort on, We set the key in this example as cust_name . We connect a flow from sort to **rollup** component. Rollup component requires a key to group the records, In this example we set the key to rollup component as cust_name and we write a transformation code as follows

```
// While we are doing the rollup for each unique key,
// we keep the following information around:

type temporary_type =
record
  decimal("\266")  total_purchases;
  string("\266") ever_used_coupon;
end;
```

```
// The initialize function sets up the initial temporary record.

temp :: initialize(in) =
begin
  temp.total_purchases     :: 0;
  temp.ever_used_coupon :: "N";
end;


// The rollup function does the work for each record in the group
// with the same key.

out :: rollup(temp, in) =
begin

temp.total_purchases :: temp.total_purchases + in.purchase;
temp.ever_used_coupon :1: if ( temp.ever_used_coupon == "Y") "Y";
temp.ever_used_coupon :2: in.ever_used_coupon;

end;


// The finalize function produces the output record from the temporary
// record and the last input record in the group.

out :: finalize(temp, in) =
begin

  out.cust_name :: in.cust_name;
  out.ever_used_coupon :: in.ever_used_coupon;
  out.total_purchases :: in.total_purchases;

end
```

Rollup component reads one record at a time from sort and compare current cust_name to next cust_name , if they are same then Rollup *function in above XFR does the work for each record in the group*. The important thing to understand here is rollup operates on each group. Every record in the group loops through rollup function in above XFR.

JOIN

Join performs inner, outer joins with multiple input datasets.

There 3 types in Ab Initio

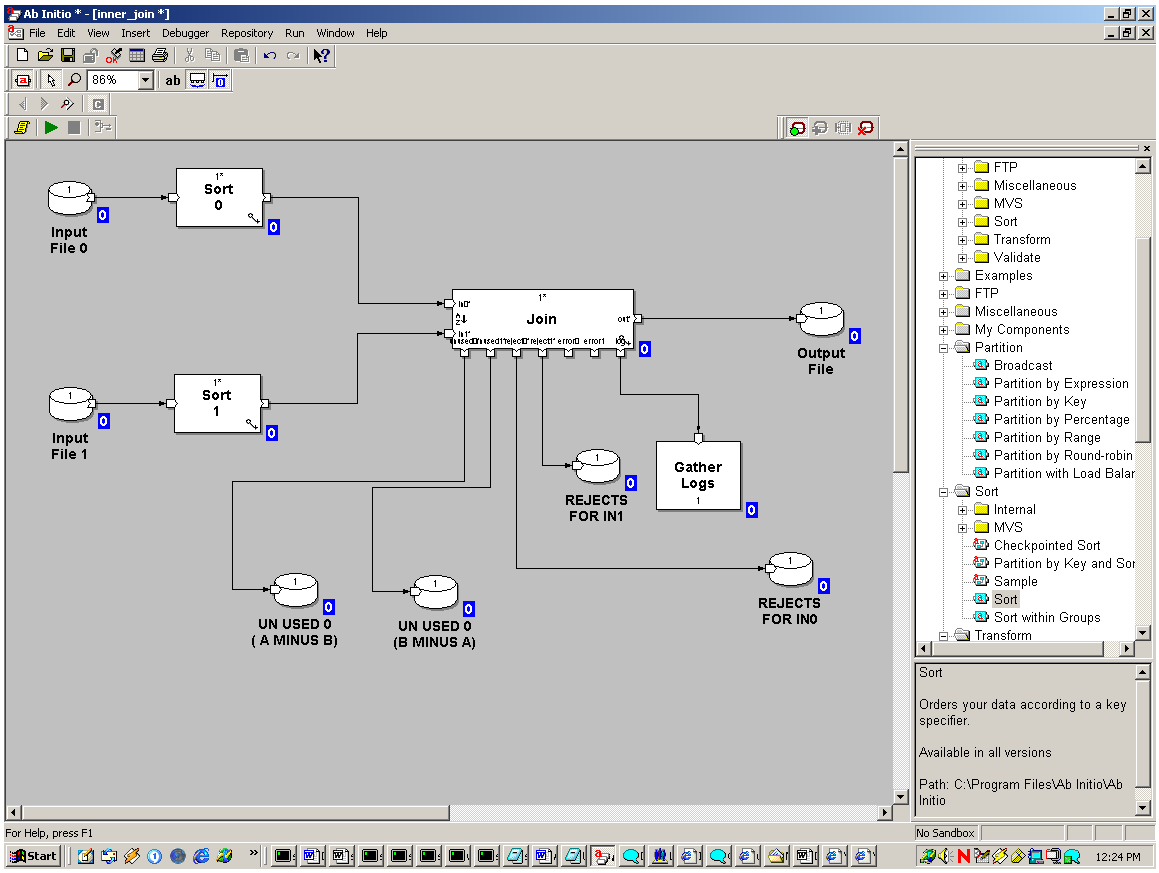
1. Inner Join, which is default
2. Explicit Join, Which is again divided into to **left outer join** and **right outer join**
3. Full outer Join.

All the joins require a Key, if you are joining two tables then the name of the Joining key in DML must be same, If not you have to use over ride key option in side the join.
If there is no key then it is called cartition join, you can do this by setting the key value to {}.

You can use unused ports to achieve A MINUS B and B MINUS A on any two files.

Joins can be done in memory by setting In-Memory option of Join. When you do In-Memory option you have to set the driving table. Join loads all the tables into memory except the driving table and performs the join in memory.

See The Example of Graph below and notice various components


Ab Initio * - [inner_join *]
File Edit View Insert Debugger Repository Run Window Help
Input File 0
Sort 0
Input File 1
Sort 1
Join
Output File
Gather Logs
REJECTS FOR IN1
REJECTS FOR IN0
UN USED 0 (A MINUS B)
UN USED 0 (B MINUS A)
FTP
Miscellaneous
MVS
Sort
Transform
Validate
Examples
FTP
Miscellaneous
My Components
Partition
Broadcast
Partition by Expression
Partition by Key
Partition by Percentage
Partition by Range
Partition by Round-robin
Partition with Load Balar
Sort
Internal
MVS
Checkpointed Sort
Partition by Key and Sor
Sample
Sort
Sort within Groups
Transform
Sort
Orders your data according to a key specifier.
Available in all versions
Path: C:\Program Files\Ab Initio\Ab Initio
For Help, press F1
No Sandbox
Start
12:24 PM

MFS AND Parallelism

Parallelism:-

There Are 3 types

1. **Component**
2. **Pipeline**
3. **Data**

Component

A graph running simultaneously on separate data using different components like Filter, Rollup, Join etc in same phase is called Component Parallelism.

Pipeline

Each component in the pipeline continuously reads from upstream components, processes data, and writes to downstream components. Since a downstream component can process records previously written by an upstream component, both components can operate in parallel.

Component and Pipeline Parallelisms are default in Ab Initio, Programmer don't have any control on these Parallelisms.

Data

Data Parallelism is achieved using Multi File System (MFS). A multifile is a parallel file that is composed of individual files on different disks and/or nodes. The individual files are partitions of the multifile. Each multifile contains one **control partition** and one or more **data partitions**. Control partition will have pointers to data partition.

If there 4 data partition then MFS is called 4 Way MFS
If you have 8 data partition then that MFS is called 8 Way MFS and so on.

Multi File System (MFS) is created using a command called **m_mkfs ,** And deleted using m_rmfs

Following command outlines how to create a MFS

m_mkfs **//razzle/data/abwork/traing/b1/my_4way** \
//razzle/data/abwork/traing/b1/d1 \
//razzle/data/abwork/traing/b1/d2 \
//razzle/data/abwork/traing/b1/d3 \
//razzle/data/abwork/traing/b1/d4 \

cd **/data/abwork/traing/b1/**

chmod 777 **my_4way**
chmod 777 **my_4way** /.WORK

m_touch **my_4way**/x.txt

The First line MFS is called control partition, you write all you files in control partition by specify your output file name in output file compoent.

To send a single file to Multifile we have to use partitioning components. There are various partitioning components.

1. **Partition by Key** :- distributes data records to its output flow partitions according to key values.
2. **Partition by Expression :-** distributes data records to its output flow partitions according to a specified DML expression.
3. **Partition by Percentage:-** distributes a specified percentage of the total number of input data records to each output flow.
4. **Partition by Range :-** distributes data records to its output flow partitions according to the ranges of key values specified for each partition.
5. **Partition by Round-robin :-** distributes data records evenly to each output flow in round-robin fashion.
6. **Partition with Load Balance:-** distributes data records to its output flow partitions, writing more records to the flow partitions that consume records faster.
7. **Broadcast: -** Broadcast can act like replicate but it does more than replicate. Boadcast can be used to send single file into MFS with out splitting. I.e if you broadcast small file with 10 records in to 4 way, Broadcast send 1 copy of 10 records to all 4 data partitions.

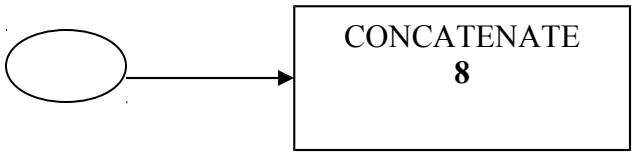
To convert Multifile into single file we have to use Departitioning components. There are various Departition components.

1. **Concatenate** : appends multiple flow partitions of data records one after another.
2. **Gather**: combines data records from multiple flow partitions arbitrarily.
3. **Interleave** : combines blocks of data records from multiple flow partitions in round-robin fashion.
4. **Merge** : combines data records from multiple flow partitions that have been sorted according to the same key specifier and maintains the sort order.

Ab Initio PERFORMANCE

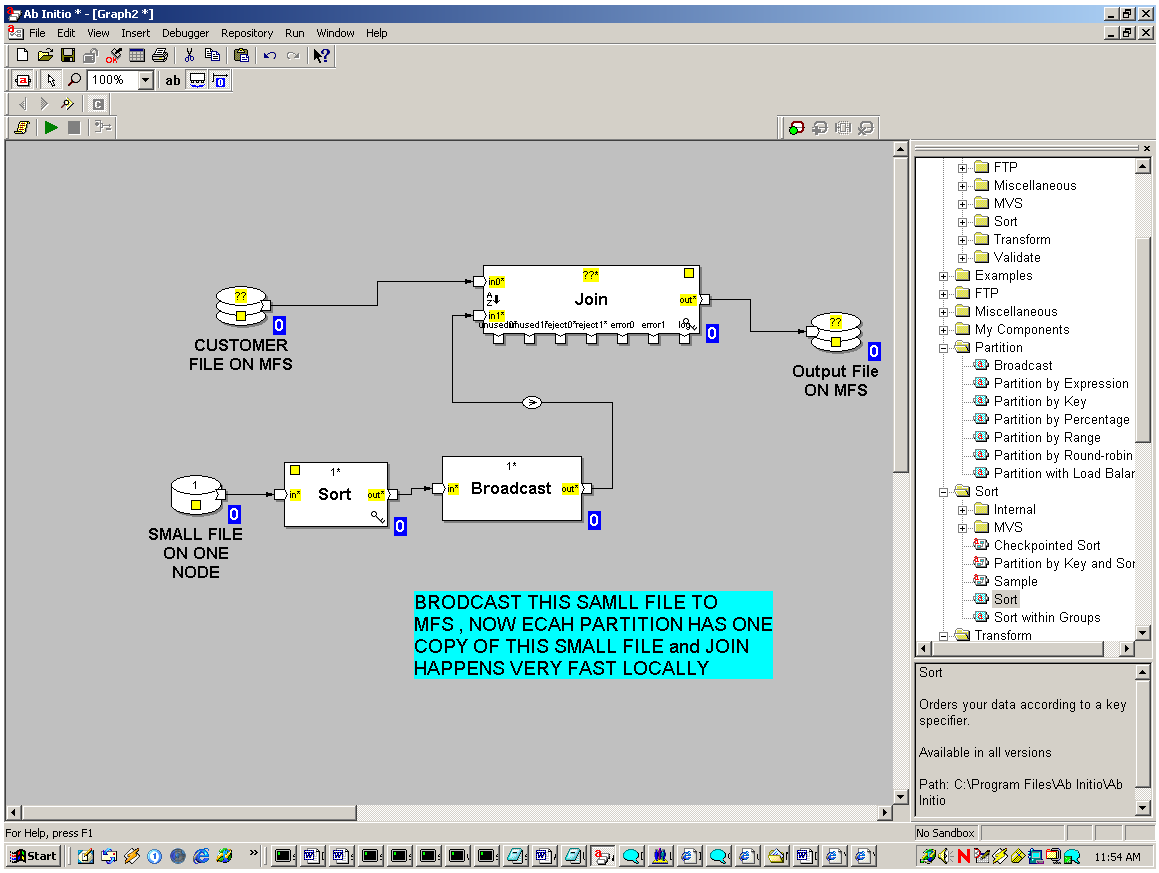
How To Improve Performance: -

1. Go Parallel as soon as possible using Ab Initio Partitioning technique.
2. Once Data Is partitioned do not bring to serial , then back to parallel. Repartition instead.
3. For Small processing jobs serial may be better than parallel.
4. Do not access large files across NFS, Use FTP component
5. Use Ad Hoc MFS to read many serial files in parallel and use concat coponenet.



Ad Hoc MFS(80 files)

1. Using Phase breaks let you allocate more memory to individual component and make your graph run faster
2. Use Checkpoint after the sort than land data on to disk
3. Use Join and rollup in-memory feature
4. Best performance will be gained when components can work with in memory by MAX-CORE.
5. MAR-CORE for SORT is calculated by finding size of input data file.
6. For In-memory join memory needed is equal to non-driving data size + overhead.
7. If in-memory join cannot fir its non-driving inputs in the provided MAX-CORE then it will drop all the inputs to disk and in-memory does not make sence.
8. Use rollup and Filter by EX as soon as possible to reduce number of records.
9. When joining very small dataset to a very large dataset, it is more efficient to broadcast the small dataset to MFS using broadcast component or use the small file as lookup.



10. Reduce number of components may save startup costs.
11. Don't use MFS if you have small datasets
12. Use select filter inside the component than separate Filter By Ex component
13. Monitor UNIX CPU usage by using **vmstat** , disk usage using **iostat .**