

Abinitio Session 1- 11, UNIX

Data ware housing

- ❖ Inmon Definition : A warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process
- ❖ Ab-Initio is a data processing tool that helps in developing Business Applications from simple to most complex
 - Data warehousing
 - Batch processing
 - Data movement
 - Data transformation

❖ Subject Oriented:

Data that gives information about a particular subject instead of about a company's ongoing operations.

❖ Integrated:

Data that is gathered into the data warehouse from a variety of sources and merged into a coherent whole.

❖ Time-variant:

All data in the data warehouse is identified with a particular time period.

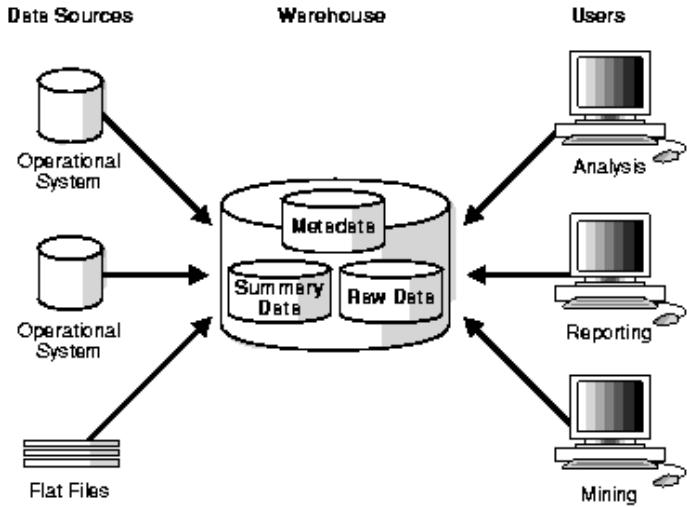
❖ Non-volatile

Data is stable in a data warehouse. More data is added but data is never removed. This enables management to gain a consistent picture of the business.

Features

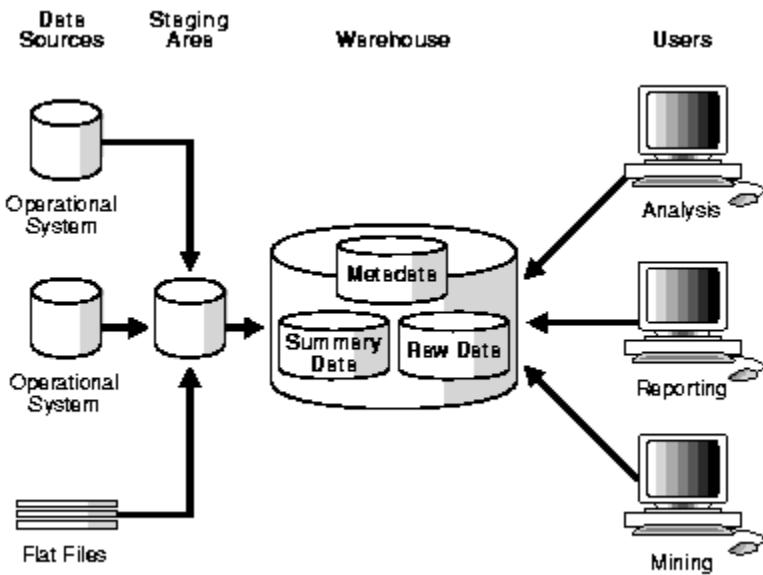
- ❖ Built-in drill-down / drill-around capability that allows detailed examination of data that comprise a summary value in a query result
- ❖ Built-in graphic capability that allows results to be viewed with bar, pie or line graphs
- ❖ Built-in export feature that allows data exports to other packages of your choice (e.g. spreadsheet, word processing or other favorite applications)

- ❖ A simple, intuitive interface that requires minimal training to effectively utilize
 - ❖ Pre-formatted query templates for novice users
 - ❖ Capability to save commonly used queries for quick retrieval and execution
 - ❖ Multi-level security for restricted data access
 - ❖ To use data models and/or server technologies that speed up querying and reporting and that are not appropriate for transaction processing
 - ❖ To provide an environment where a relatively small amount of knowledge of the technical aspects of database technology is required to write and maintain queries and reports and/or to provide a means to speed up the writing and maintaining of queries and reports by technical personnel
 - ❖ To provide a repository of "cleaned up" transaction processing systems data that can be reported against and that does not necessarily require fixing the transaction processing systems
 - ❖ To make it easier, on a regular basis, to query and report data from multiple transaction processing systems and/or from external data sources and/or from data that must be stored for query/report purposes only
 - ❖ To provide a repository of transaction processing system data that contains data from a longer span of time than can efficiently be held in a transaction processing system and/or to be able to generate reports "as was" as of a previous point in time
 - ❖ Data warehouses and their architectures vary depending upon the specifics of an organization's situation. Three common architectures are:
 1. Data Warehouse Architecture (Basic)
 2. Data Warehouse Architecture (with a Staging Area)
 3. Data Warehouse Architecture (with a Staging Area and Data Marts)
- ❖ **Figure 1-2 Architecture of a Data Warehouse**



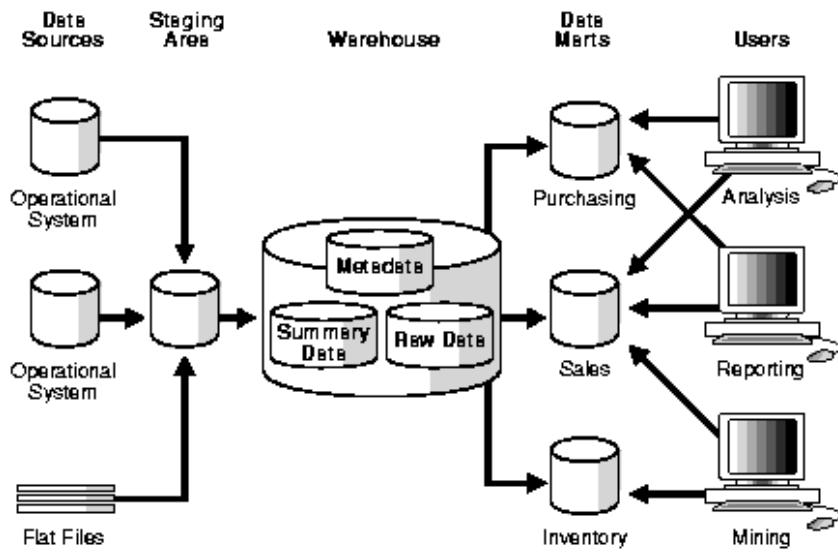
- ❖ In Figure 1-2, the metadata and raw data of a traditional OLTP system is present, as is an additional type of data, summary data. Summaries are very valuable in data warehouses because they pre-compute long operations in advance. For example, a typical data warehouse query is to retrieve something like August sales. A summary in Oracle is called a **materialized view**.

❖ ***Figure 1-3 Architecture of a Data Warehouse with a Staging Area***



In [Figure 1-2](#), you need to clean and process your operational data before putting it into the warehouse. You can do this programmatically, although most data warehouses use a **staging area** instead. A staging area simplifies building summaries and general warehouse management. [Figure 1-3](#) illustrates this typical architecture

❖ ***Figure 1-4 Architecture of a Data Warehouse with a Staging Area and Data Marts***



- ❖ Although the architecture in [Figure 1-3](#) is quite common, you may want to customize your warehouse's architecture for different groups within your organization. You can do this by adding **data marts**, which are systems designed for a particular line of business. [Figure 1-4](#) illustrates an example where purchasing, sales, and inventories are separated. In this example, a financial analyst might want to analyze historical data for purchases and sales.

DATA MODELS

- ❖ Star Schema
❖ Snow Flake Schema

Glossary:

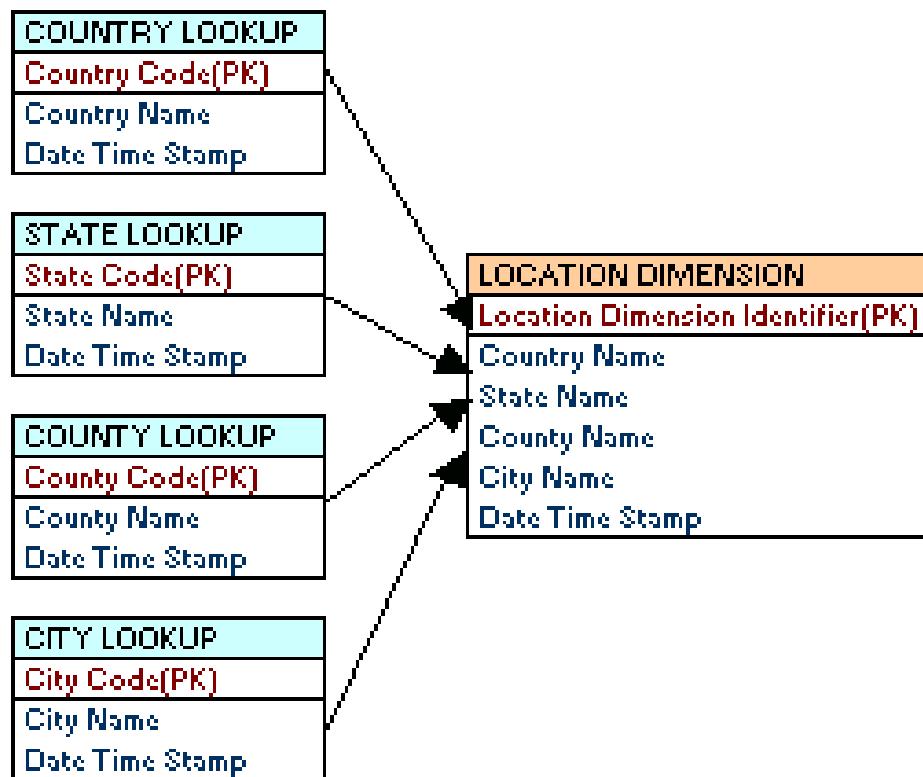
❖ **Dimension Table**

Dimension table is one that describe the business entities of an enterprise, represented as hierarchical, categorical information such as time,

departments, locations, and products. Dimension tables are sometimes called lookup or reference tables.

❖ **Location Dimension**

In a relational data modeling, for normalization purposes, country lookup, state lookup, county lookup, and city lookups are not merged as a single table. In a dimensional data modeling(star schema), these tables would be merged as a single table called LOCATION DIMENSION for performance and slicing data requirements. This location dimension helps to compare the sales in one region with another region. We may see good sales profit in one region and loss in another region. If it is a loss, the reasons for that may be a new competitor in that area, or failure of our marketing strategy etc



Star Schema is a relational database schema for representing multidimensional data. It is the simplest form of data warehouse schema that contains one or more dimensions and fact tables. It is called a star schema because the entity-relationship diagram between dimensions and fact tables resembles a star where one fact table is connected to multiple dimensions. The center of the star schema consists of a large fact table and it points towards the

dimension tables. The advantage of star schema are slicing down, performance increase and easy understanding of data.

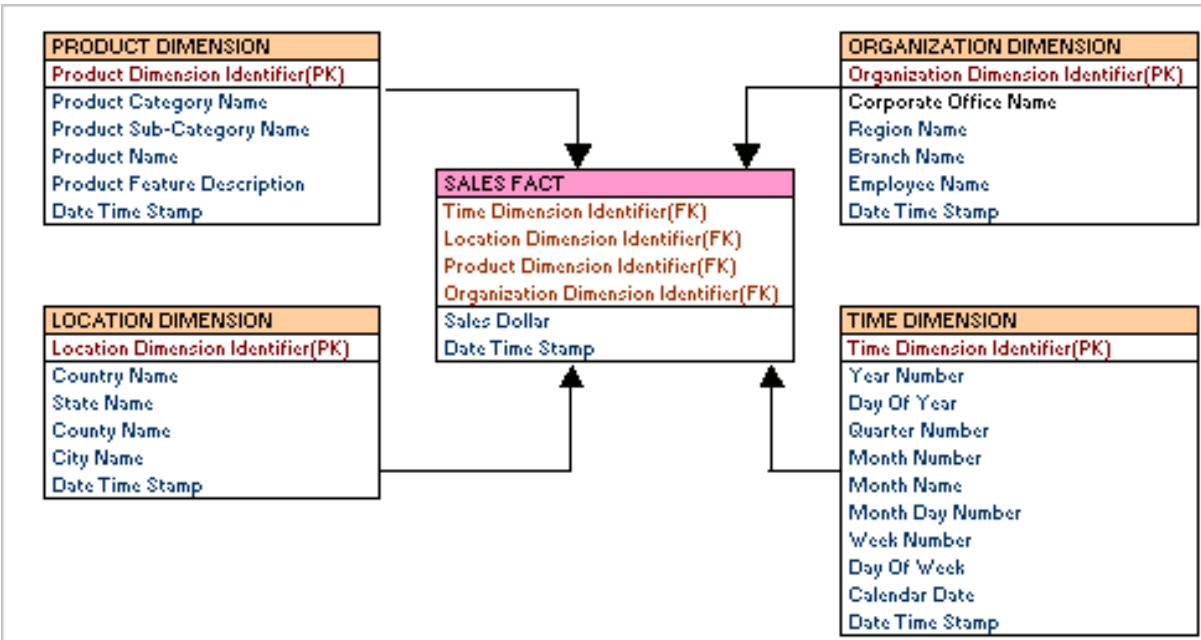
- ❖ **Steps in designing Star Schema**
- ❖ **Identify a business process for analysis(like sales).**
- ❖ **Identify measures or facts (sales dollar).**
- ❖ **Identify dimensions for facts(product dimension, location dimension, time dimension, organization dimension).**
- ❖ **List the columns that describe each dimension.(region name, branch name, region name).**
- ❖ **Determine the lowest level of summary in a fact table(sales dollar).**
- ❖ **Hierarchy**

A logical structure that uses ordered levels as a means of organizing data. A hierarchy can be used to define data aggregation; for example, in a time dimension, a hierarchy might be used to aggregate data from the Month level to the Quarter level, from the Quarter level to the Year level. A hierarchy can also be used to define a navigational drill path, regardless of whether the levels in the hierarchy represent aggregated totals or not.
- ❖ **Level**

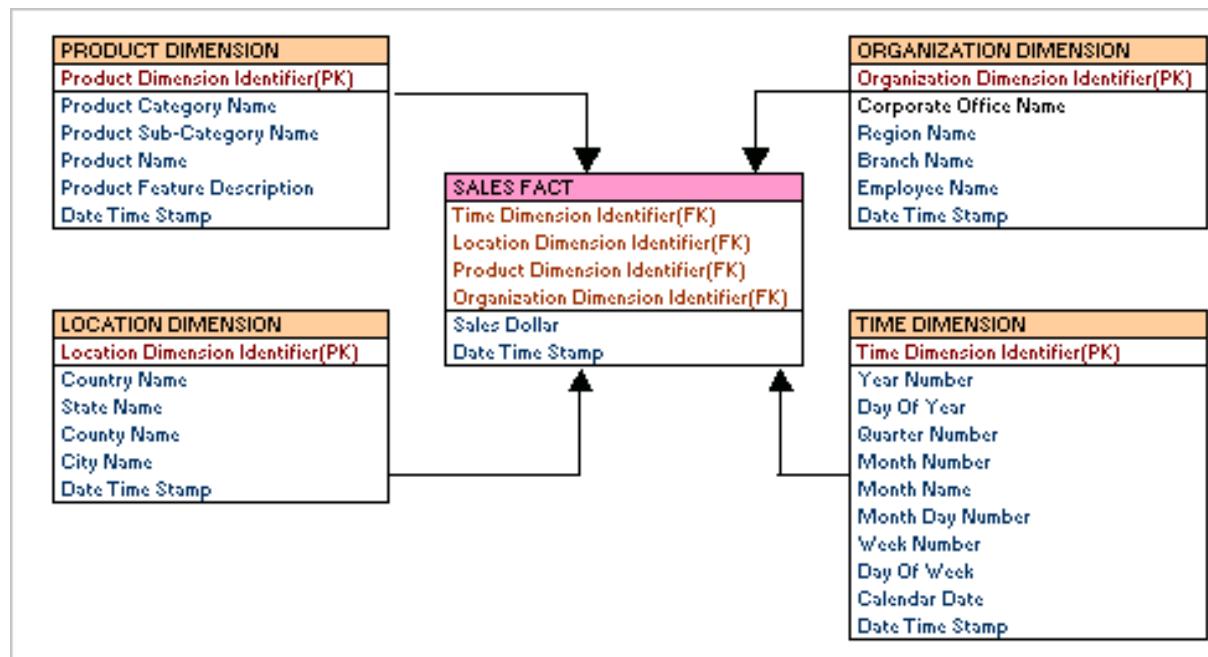
A position in a hierarchy. For example, a time dimension might have a hierarchy that represents data at the Month, Quarter, and Year levels.
- ❖ **Fact Table**

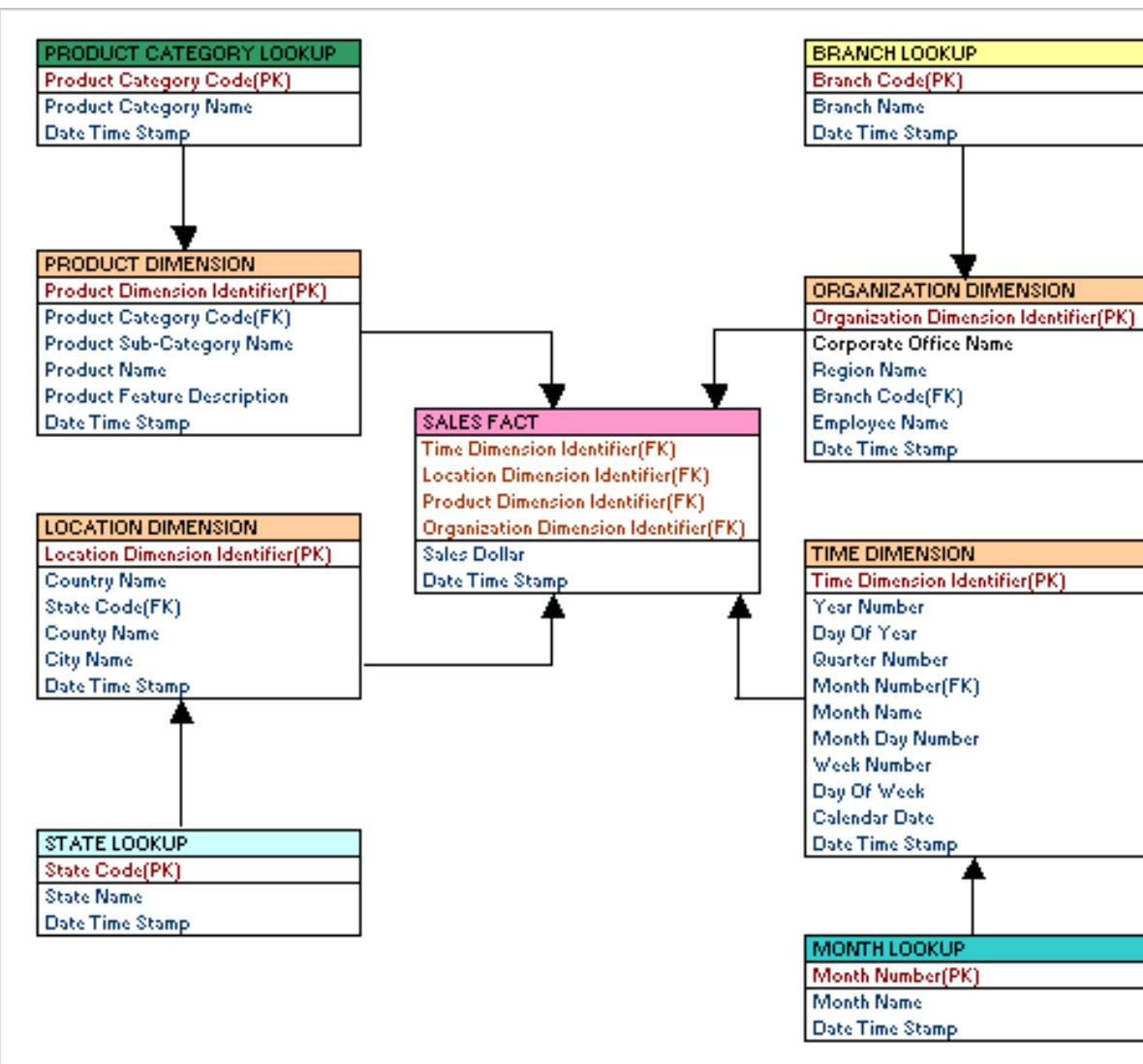
A table in a star schema that contains facts and connected to dimensions. A fact table typically has two types of columns: those that contain facts and those that are foreign keys to dimension tables. The primary key of a fact table is usually a composite key that is made up of all of its foreign keys.
- ❖ **A fact table might contain either detail level facts or facts that have been aggregated (fact tables that contain aggregated facts are often instead called summary tables). A fact table usually contains facts with the same level of aggregation.**

DWH ARCH



SNOW FLAKE SCHEMA





ETL Tools are meant to extract, transform and load the data into Data Warehouse for decision making.

❖ Source System

A database, application, file, or other storage facility from which the data in a data warehouse is derived.

❖ Mapping

The definition of the relationship and data flow between source and target objects.

❖ Metadata

Data that describes data and other structures, such as objects, business rules, and processes. For example, the schema design of a data warehouse is typically stored in a repository as metadata, which is used to generate scripts used to build and populate the data warehouse. A repository contains metadata.

❖ Staging Area

A place where data is processed before entering the warehouse.

❖ Cleansing

The process of resolving inconsistencies and fixing the anomalies in source data, typically as part of the ETL process.

❖ Transformation

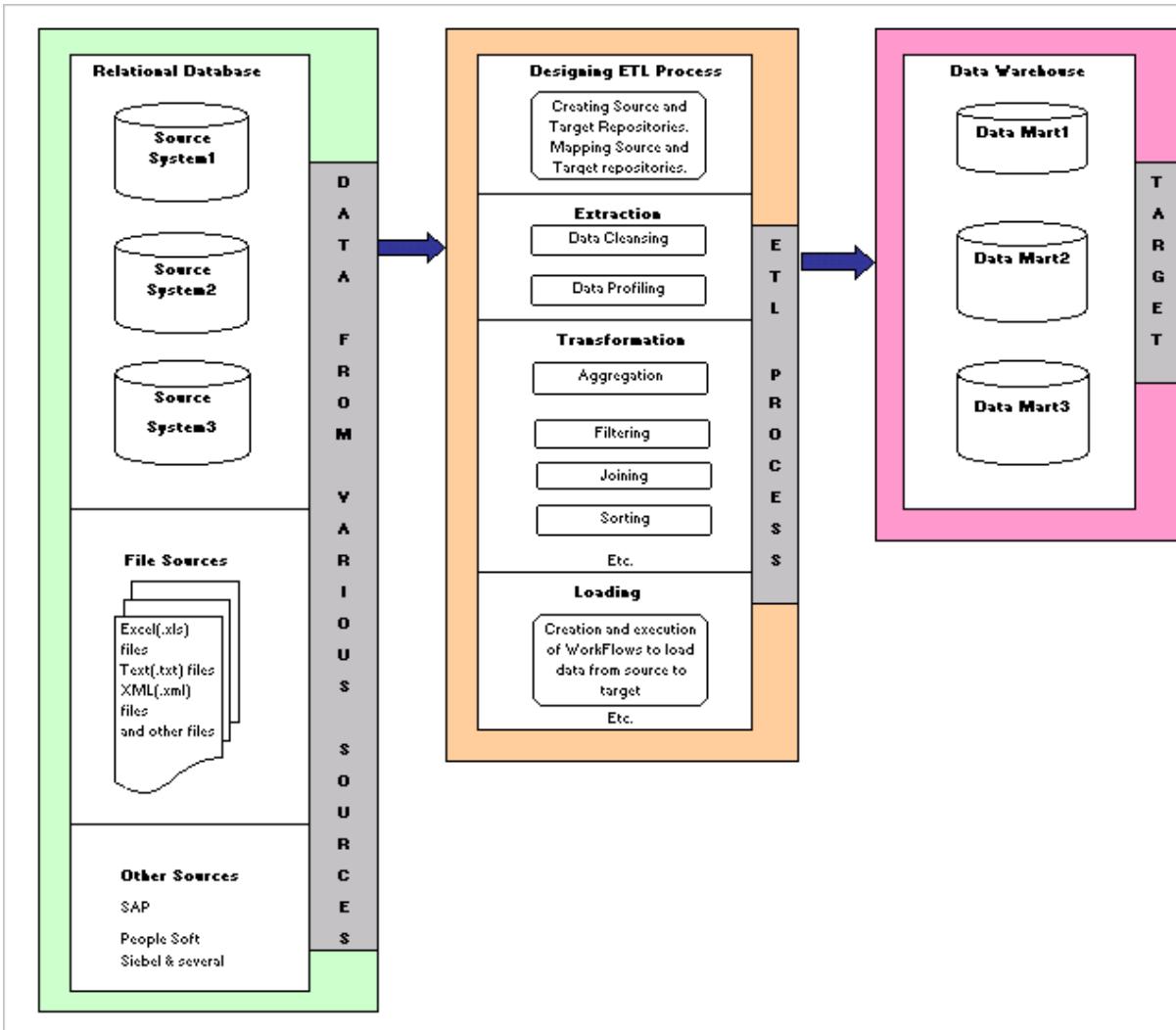
The process of manipulating data. Any manipulation beyond copying is a transformation. Examples include cleansing, aggregating, and integrating data from multiple sources.

❖ Transportation

The process of moving copied or transformed data from a source to a data warehouse.

❖ Target System

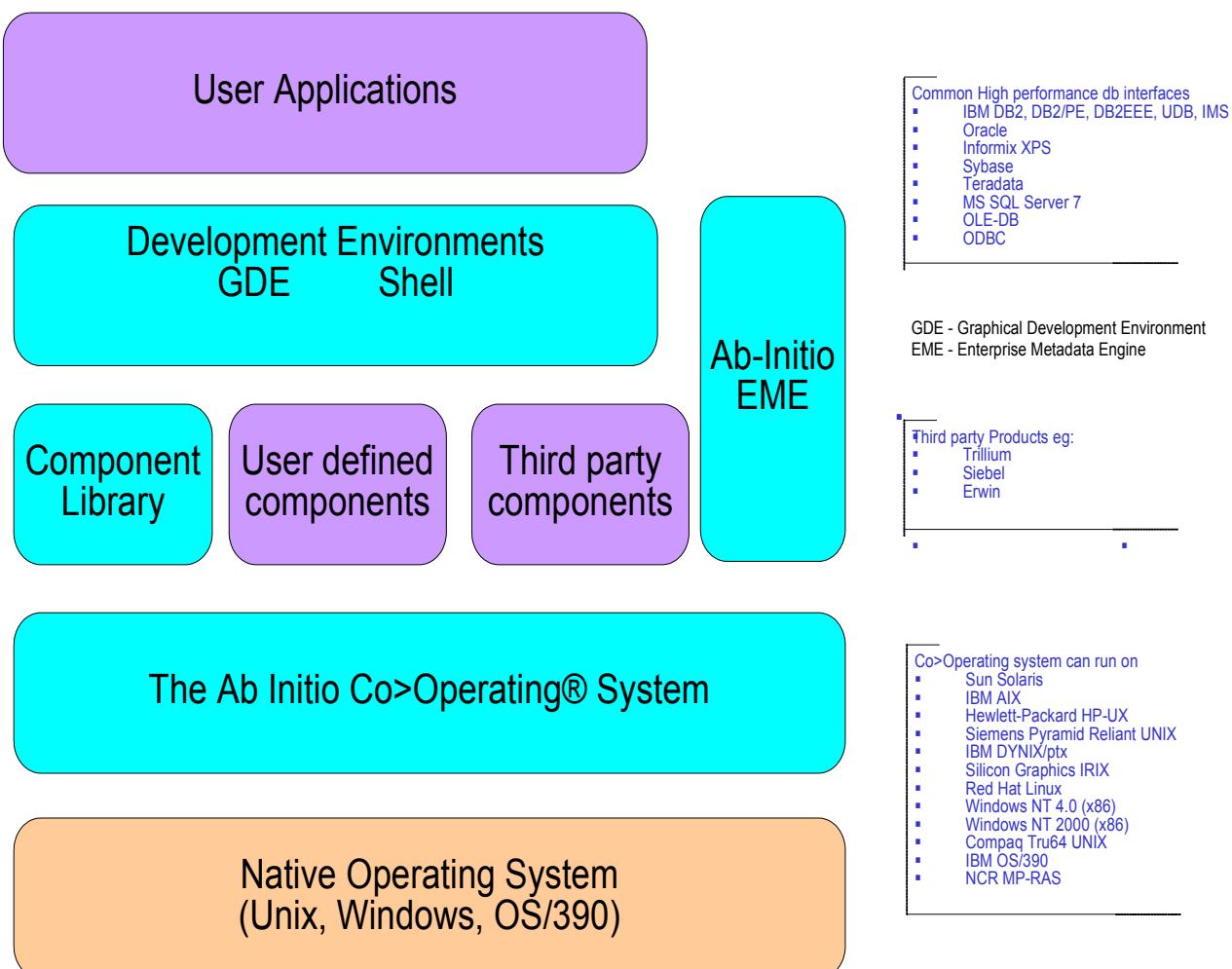
A database, application, file, or other storage facility to which the "transformed source data" is loaded in a data warehouse.



ABINITIO INTRO

- ❖ Ab Initio is Latin for “From the Beginning.”
- ❖ Ab-Initio is a data processing tool that helps in developing Business Applications from simple to most complex
 - Data warehousing
 - Batch processing
 - Data movement
 - Data transformation
- ❖ Easy Development using GDE (Graphical Development environment)

- ❖ Efficient & quick data processing
 - move small and large volumes of data in an efficient manner
 - deal with the complexity associated with business data
- ❖ High Performance
 - scalable solutions
- ❖ Compatibility with various data sources
- ❖ Available in both NT & Unix
- ❖ Checkpoint and restartability
- ❖ Better productivity



Client PC

GDE is installed

For Development & Testing

Need Control Node to Function

MOUNT POINT

- ❖ In a Unix environment, the mount command attaches discs, or directories logically rather than physically. The Unix mount command makes a directory accessible by attaching a root directory of one file system to another directory, which makes all the file systems usable as if they were subdirectories of the file system they are attached to. Unix recognizes devices by their location, as compared to Windows, which recognizes them by their names (C: drive, for example). Unix organizes directories in a tree-like structure, in which directories are attached by *mounting* them on the branches of the tree. The file system location where the device is attached is called a *mount point*.

CO-OP SERVICES

- ❖ Parallel and distributed application execution
 - Control

- Data Transport
- ❖ Checkpoint
- ❖ Monitoring and debugging.
- ❖ Parallel file management.
- ❖ Metadata-driven components

Port

- ❖ **DML**
 - **Ab Initio stores metadata in the form of record formats.**
 - **Metadata can be embedded within a component or can be stored external to the graph in a file with a “.dml” extension.**
- ❖ **XFR**
 - **Data can be transformed with the help of transform functions.**
 - **Transform functions can be embedded within a component or can be stored external to the graph in a file with a “.xfr” extension.**

Data Types

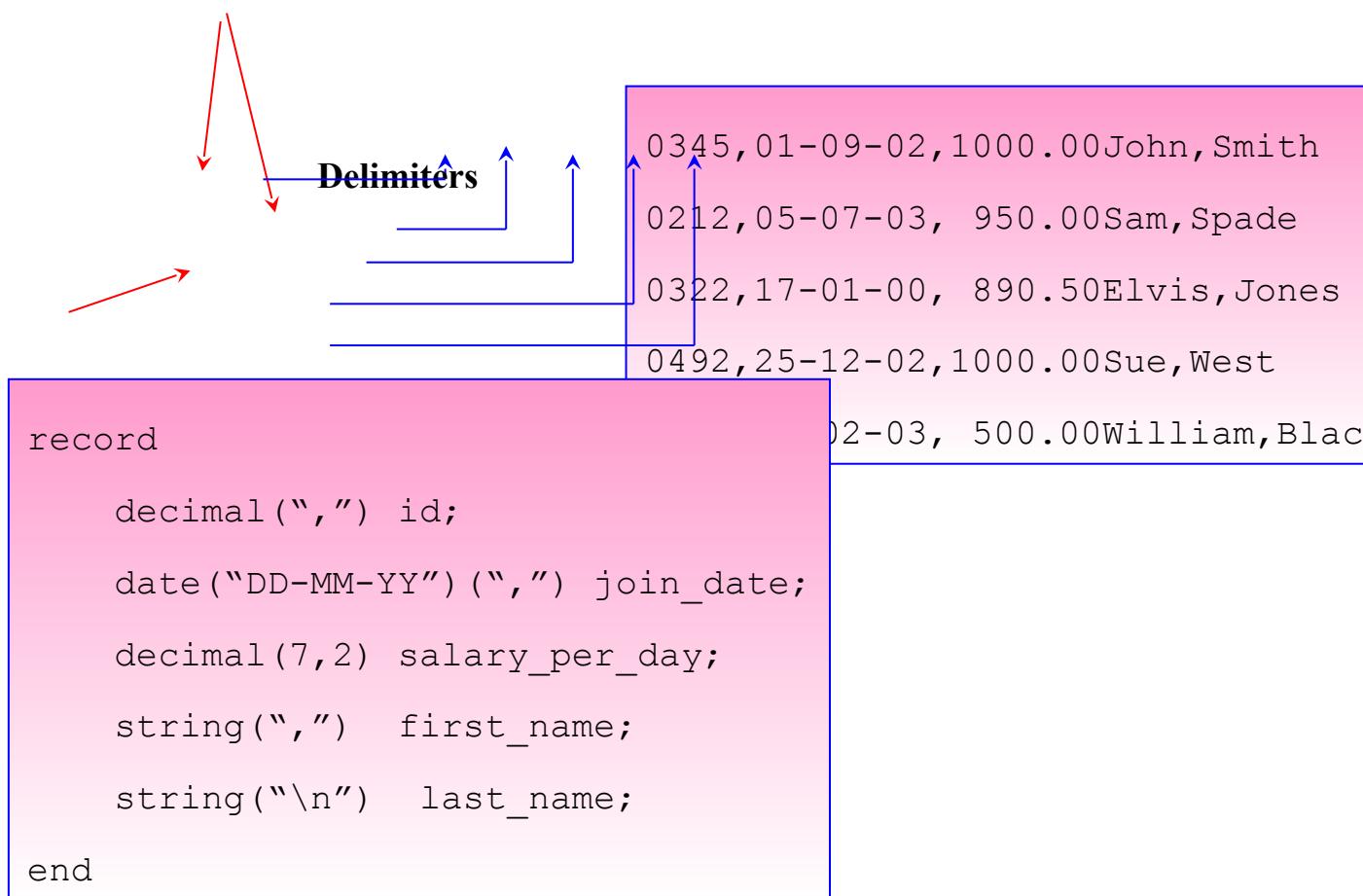
DML Syntax

- **Record types begin with *record* and end with *end***
- **Fields are declared: *data_type(length) field_name;* (fixed length DML)**

data_type(delimiter) field_name; (delimited DML)
- **Field names consist of letters(a...z,A...Z), digits(0...9), underscores(_) and are Case sensitive**
- **Keywords/Reserved words are *record, end, date....***

Data Types

- **String**
- **Decimal**
- **Integer**
 - ✧ **Storing Data in binary form**
- **Date and Datetime**
- **EBCDIC and ASCII records**
-



Records

Table Data

Header

Header **NULL**

NULL

Body

NULL Body **NULL**

Trailer

NULL **NULL** Trailer

Is_DEFINED

The **is_defined** function returns:

The value **1** if *expr* evaluates to a non-NULL value. The value **0** otherwise.

Eg:

- `is_defined(123) => 1`
- `is_defined(NULL) => 0`

XFR

- **Transform functions are a collection of business rules, local variables and statements**
- **Associated with transform components**
- **Rules that computes expression from input values and local variable and assigns the result to output objects**
- **Syntax**

◊ **Functions :**

output-records : : function-name (input-records) =

begin

assignments

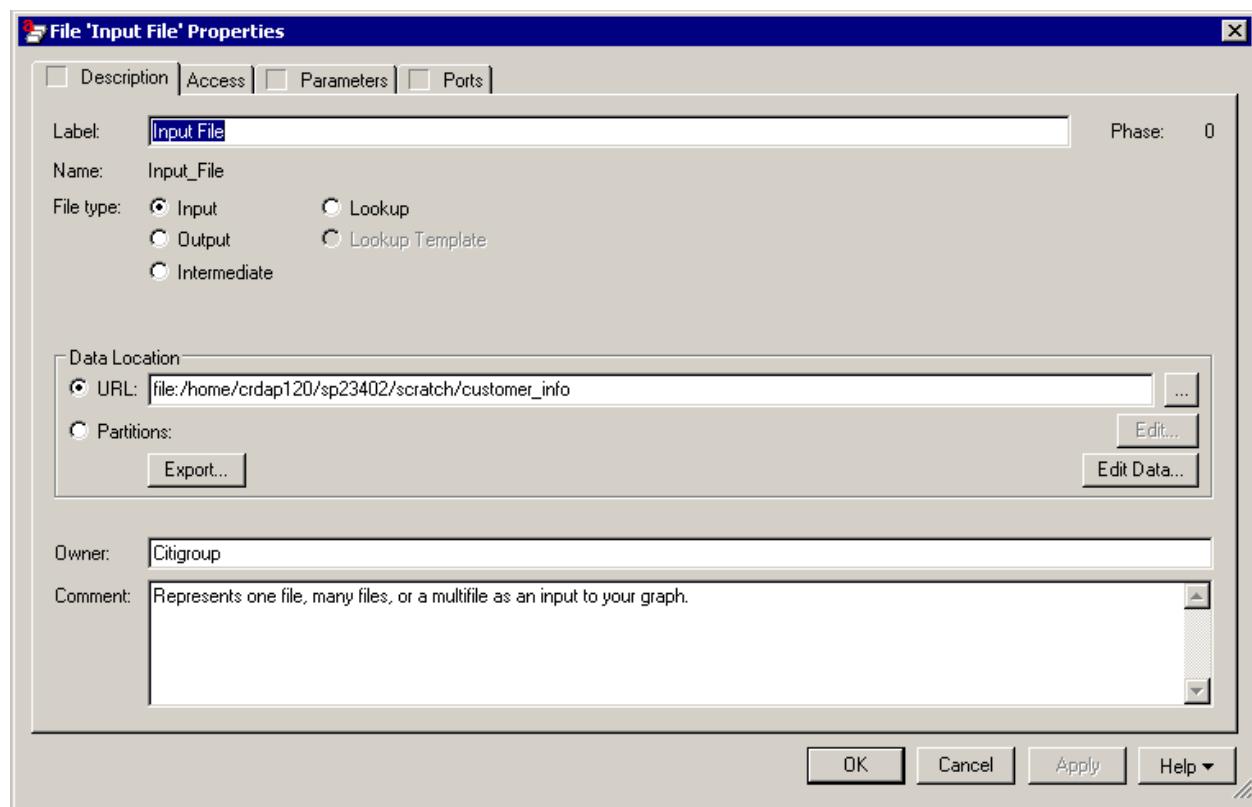
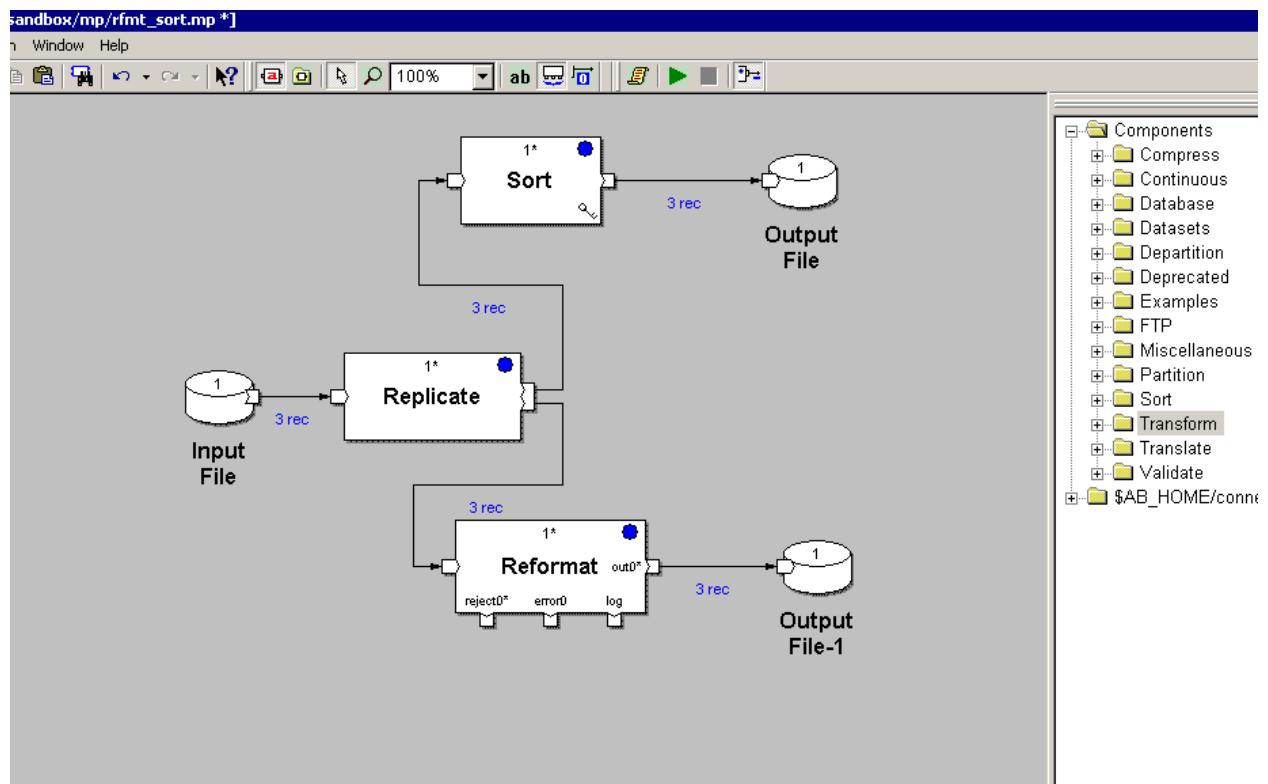
end;

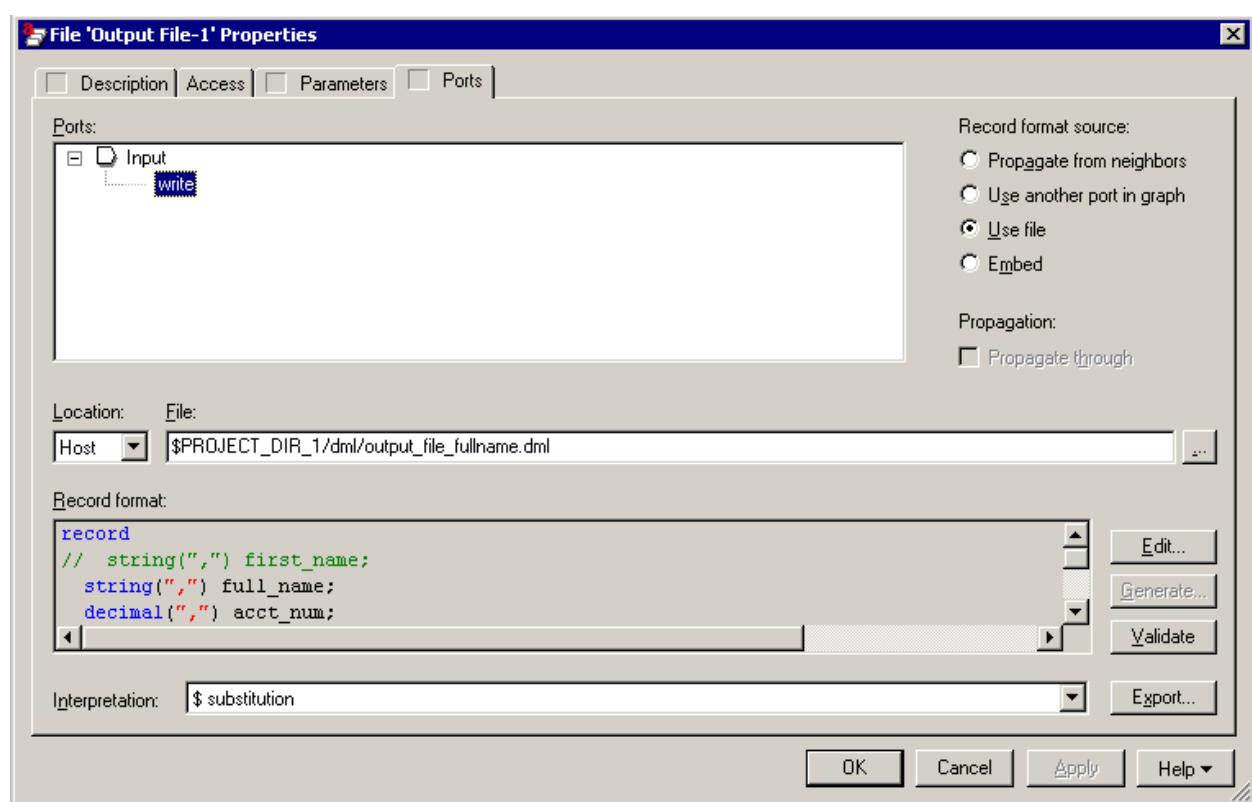
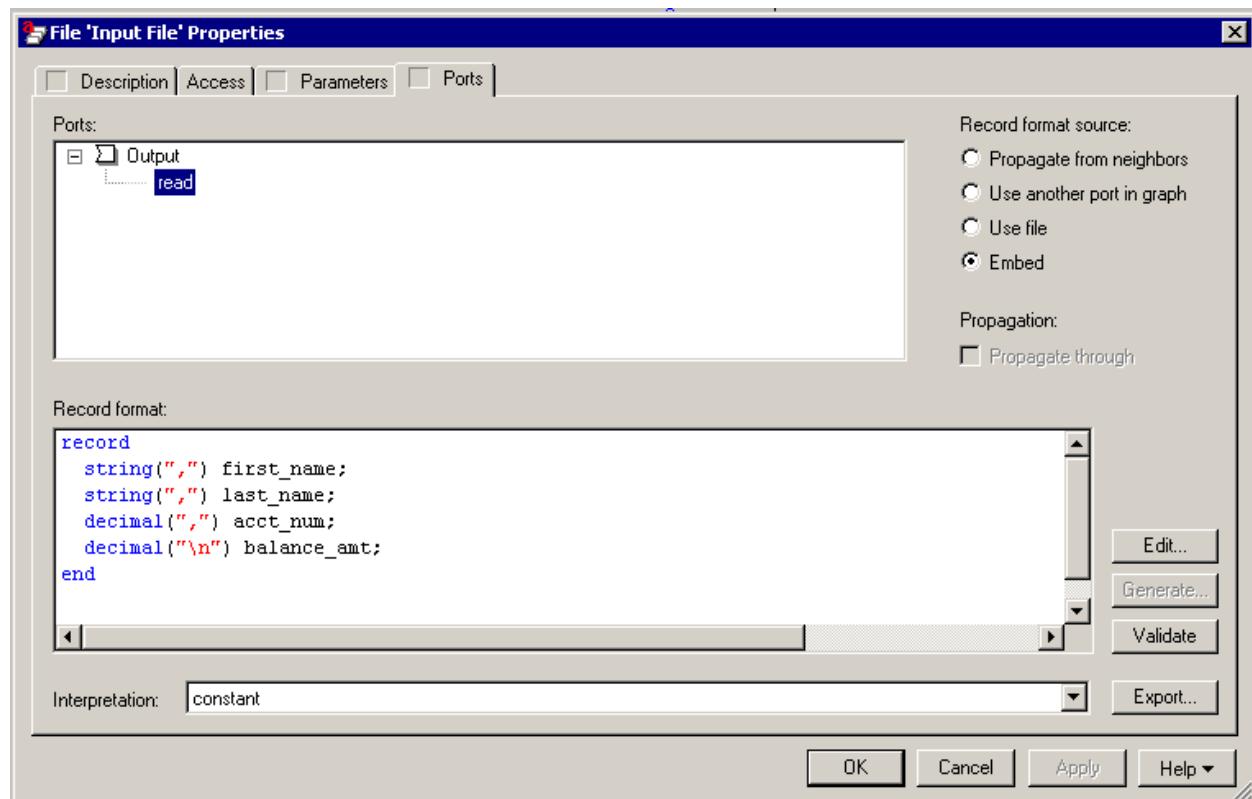
✧ **Assignments :**

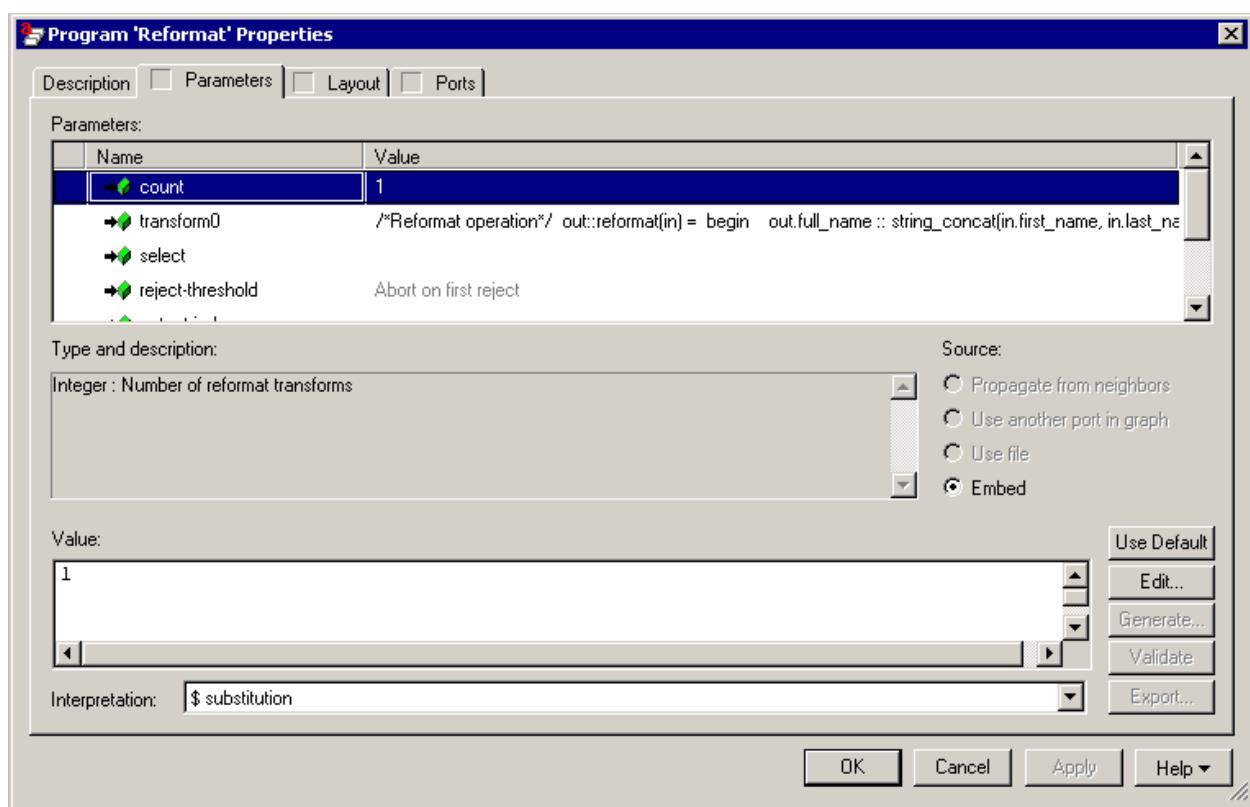
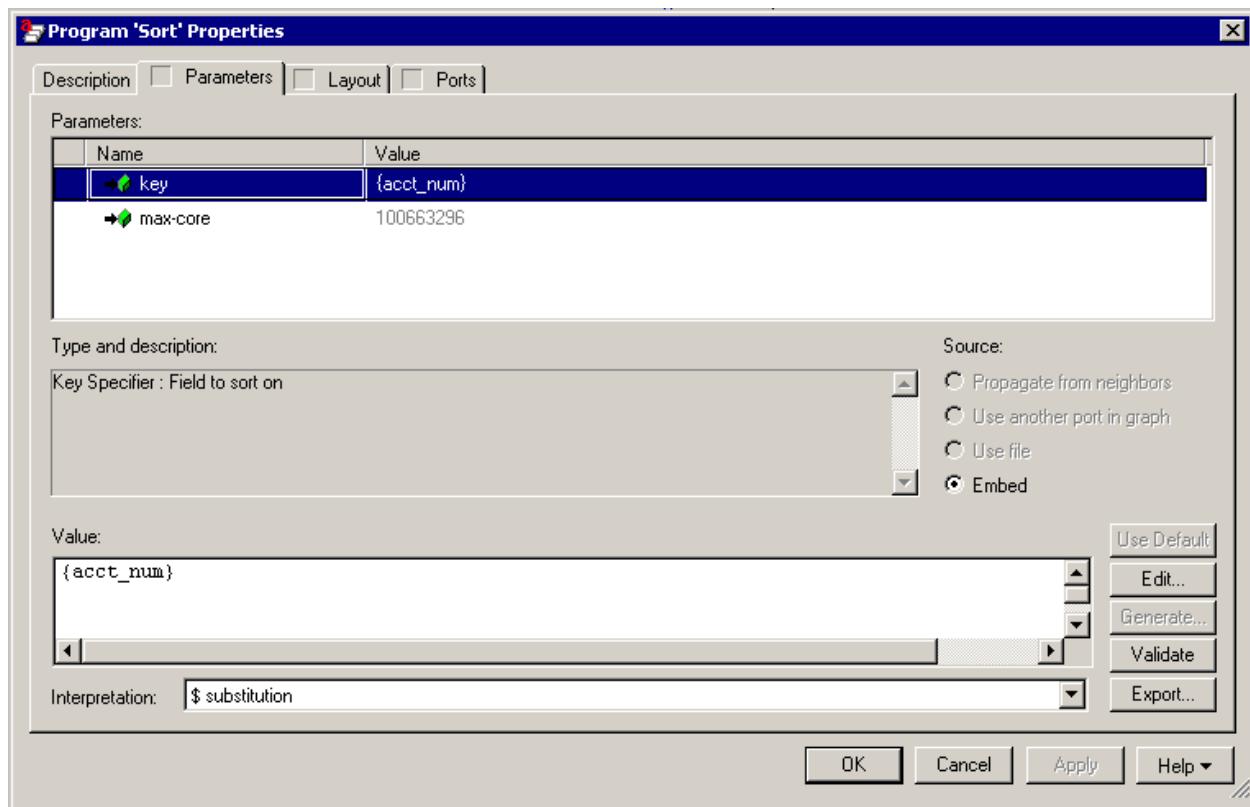
output-records.field : : expression;

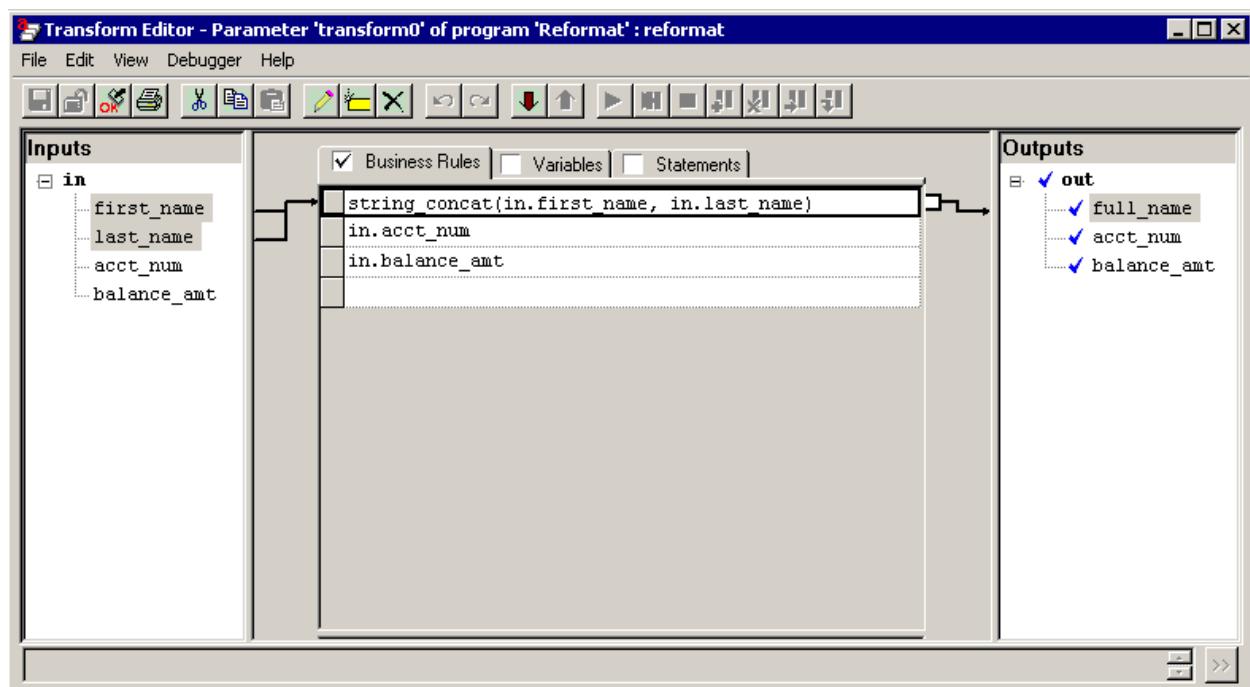
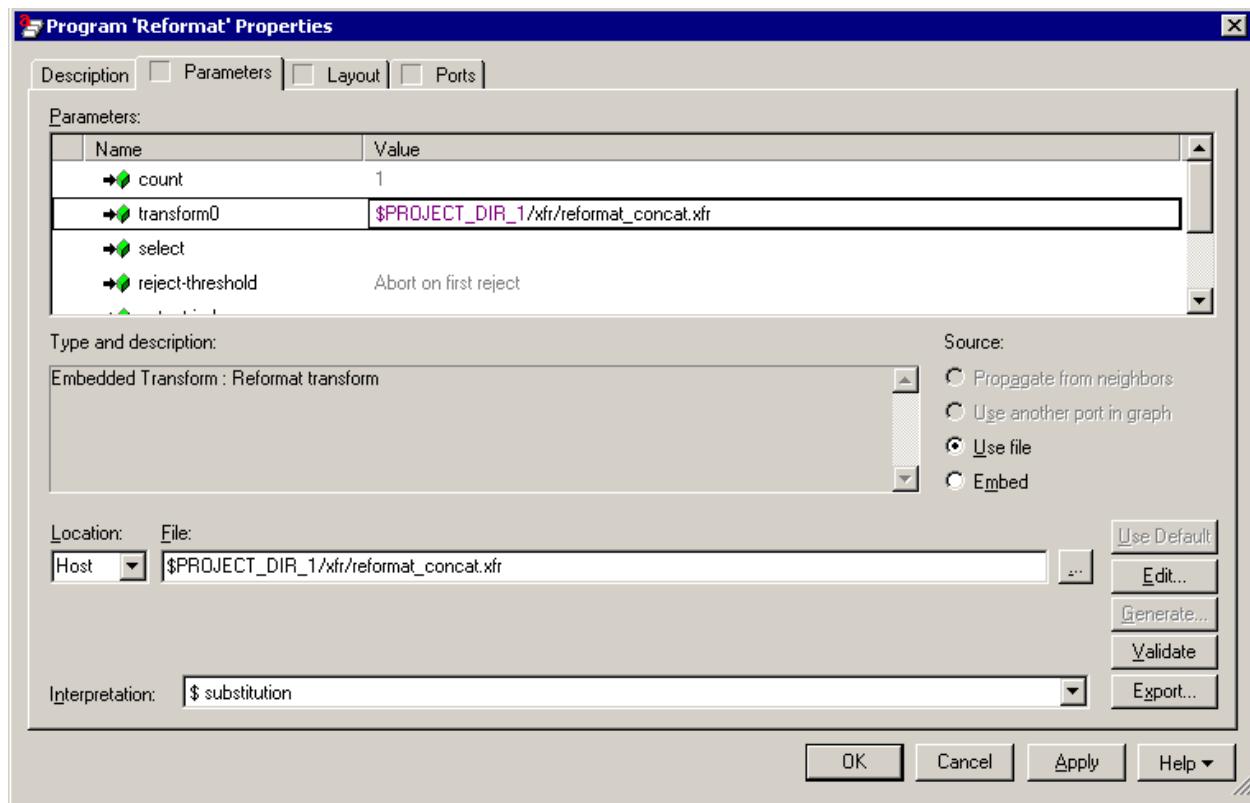
BASIC COMPONENTS

- **Reformat**
- **Sort**
- **Replicate**
- **Compress**
- **Uncompress**
- **Inflate**
- **Deflate**
- **Filter by Expression**
- **Dedup Sorted**









The screenshot shows a software interface titled "Transform Editor - Parameter 'transform0' of program 'Reformat'". The menu bar includes "File", "Edit", "View", "Debugger", and "Help". Below the menu is a toolbar with various icons. The main area contains a code editor with the following C++ code:

```
01 /*Reformat operation*/
02 out::reformat(in) =
03 begin
04     out.full_name :: string_concat(in.first_name, in.last_name);
05     out.acct_num :: in.acct_num;
06     out.balance_amt :: in.balance_amt;
07 end;
```

- **Compress** reduces the volume of data in a flow.
- The Compress component uses the Unix **compress** utility. On Windows platforms, Compress is not available. On Linux platforms, Compress cannot output more than 2 GB of data.

➤ **Parameters**

None.

➤ **Runtime Behavior**

- Reads data records from the **in** port Compresses them using the Unix **compress** utility and writes the compressed records to the **out** port
- UNCOMPRESS
- **Uncompress** uncompresses data that was compressed by the Compress component or the GZip component.
- On Windows platforms, Uncompress is not available. On Linux platforms, Uncompress cannot handle files that contain more than 2 GB of data.

➤ **Parameters**

None.

➤ **Runtime Behavior**

Reads the stream of bytes produced by Compress on its **in** port Writes uncompressed records to its out port

INFATE

- **Inflate** restores compressed data. The Inflate component can uncompress data compressed by Following:
 1. The Deflate component
 2. The GZip component
 3. The freely available **gzip** utility
- The Inflate component cannot uncompress data compressed by the following. Use the Uncompress component for data compressed by Following:
 1. The Compress component
 2. The Unix **compress** utility
 3. The Unix **pack** utility

➤ **Parameters**

None

DEFLATE

- **Deflate** reduces the volume of data in a flow. Use Deflate if you :
 1. Have a narrow bandwidth connection
 2. Lack enough disk space to store the data you
need to store
 3. Want to conserve disk space
- The Inflate component cannot uncompress data compressed by the following. Use the Uncompress component for data compressed by Following:
 1. The Compress component
 2. The Unix **compress** utility

3. The Unix **pack** utility

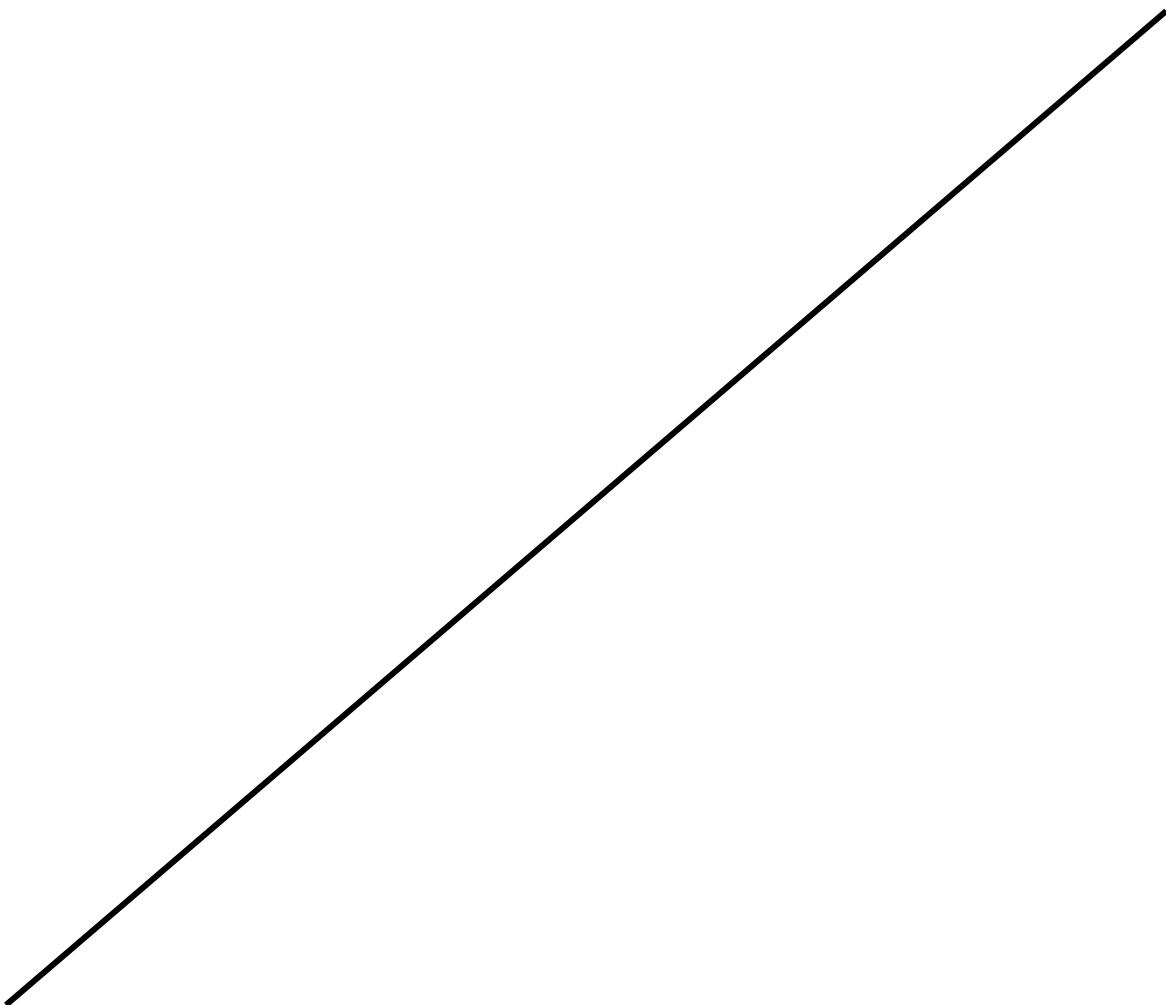
PARAMETER FOR DEFLATE

➤ **compression**

Specifies how much to compress the data. Choose from the following:

- **0** — No compression.
- **1** — Causes Deflate to run fastest, but compresses least.
- **2** through **8**— Intermediate choices
- **9** — Causes Deflate to run slowest, but compresses most.
- The default value is **1**, which causes Deflate to run fastest, but compress the least. When the value is **1**, Deflate compresses well—typically better than the Compress component — while performing much better than with **9** (the best compression).

FILTER BY EXP



DIAGNOSTIC PORTS

□ REJECT

- ◊ **Input records that caused error**

□ ERROR

- ◊ **Associated error message**

□ LOG

- ◊ **Logging records**

Note: Every transform component has got diagnostic ports

REFORMAT

1. **Reads record from *input* port**

2. Record passes as argument to *transform function or xfr*
3. Records written to *out* ports, if the function returns a success status
4. Records written to *reject* ports, if the function returns a failure status

Parameters of Reformat Component

- **Count**
- **Transform (Xfr) Function**
- **Reject-Threshold**
 - ◊ **Abort**
 - ◊ **Never Abort**
 - ◊ **Use Limit & Ramp**
 - ⇒ **Limit**
 - ⇒ **Ramp**

INSTRUMENTATION PARAMETERS

- **Limit**
 - ◊ **Number of errors to tolerate**
- **Ramp**
 - ◊ **Scale of errors to tolerate per input**
- **Tolerance value=limit + ramp*total number of records read**
- **Typical Limit and Ramp settings . .**
 - ◊ **Limit = 0 Ramp = 0.0 □ Abort on any error**
 - ◊ **Limit = 50 Ramp = 0.0 □ Abort after 50 errors**
 - ◊ **Limit = 1 Ramp = 0.01 □ Abort if more than 1 in 100 records causes error**
 - ◊ **Limit = 1 Ramp = 1 □ Never Abort**

SORT

Key

A key identifies a field or set of fields to organize a dataset

- ✧ **Single Field:** *employee_number*
- ✧ **Multiple field or Composite key:** (*last_name; first_name*)
- ✧ **Modifiers:** *employee_number descending*

A surrogate key is a substitution for the natural primary key.

- ⇒ **It is just a unique identifier or number for each record like ROWID of an Oracle table**

Sort Component :

- **Reads records from input port, sorts them by key, writes result to output port**
- **Parameters**
 - ✧ **Key**
 - ✧ **Max-core**

JOIN

Reads records from multiple input ports

Operates on records with matching keys using a multi-input transform function

Writes result to the output port

DEDUP SORTED

- Separates one specified record in each group of records from the rest of the records in the group
- Requires sorted input

REPLICATE

- Arbitrarily combines the records from all the flows on the in port into a single flow
- Copies that flow to all the flows connected to the out port

MULTISTAGE TRANSFORM

- **Data transformation in multiple stages following several sets of rules**
- **Each set of rule form one transform function**
- **Information is passed across stages by temporary variables**

- **Stages include initialization, iteration, finalization and more**
- **Few multistage components are aggregate, rollup, scan**

Rollup

Aggregate

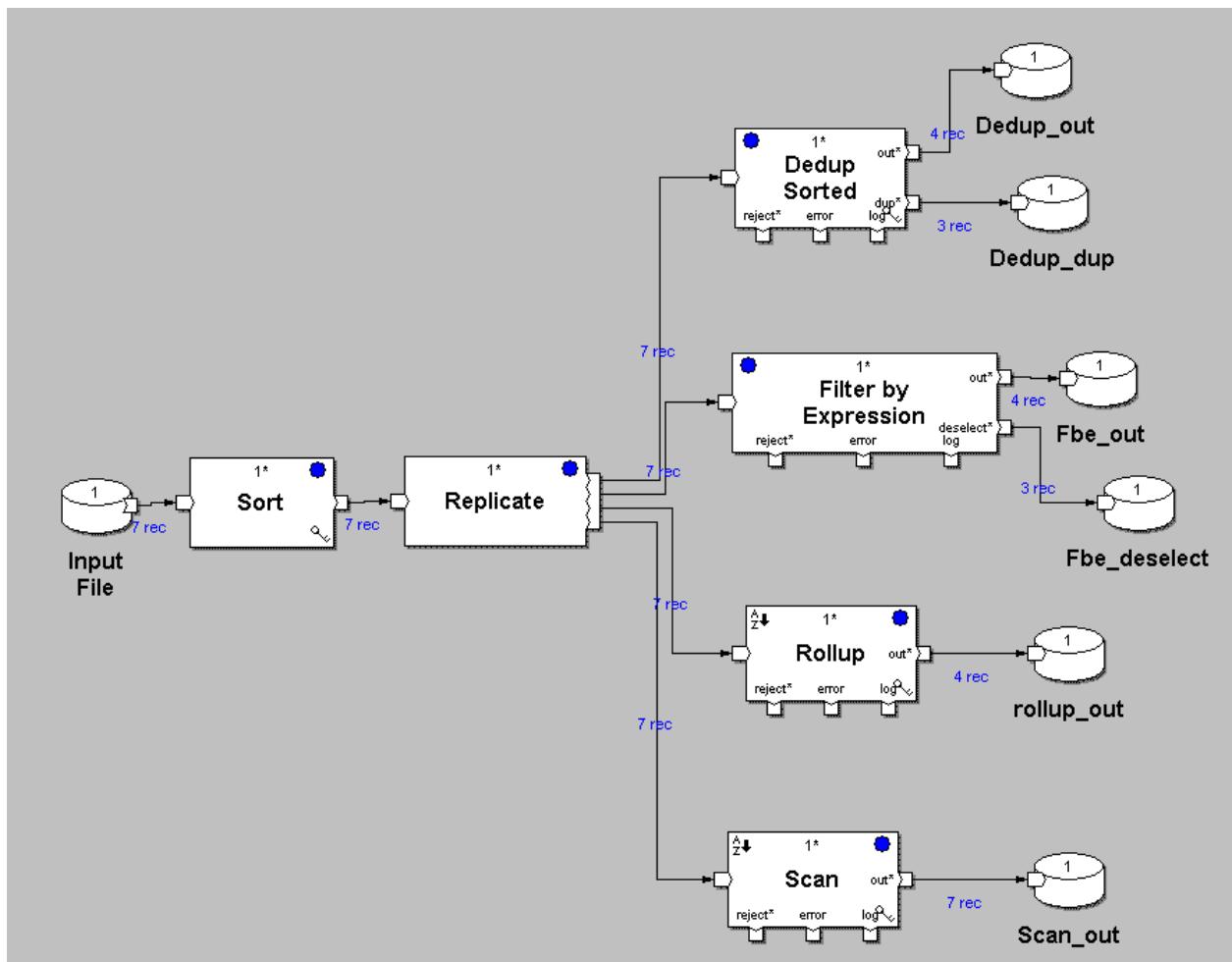
Scan

Generates summary records for group of input records

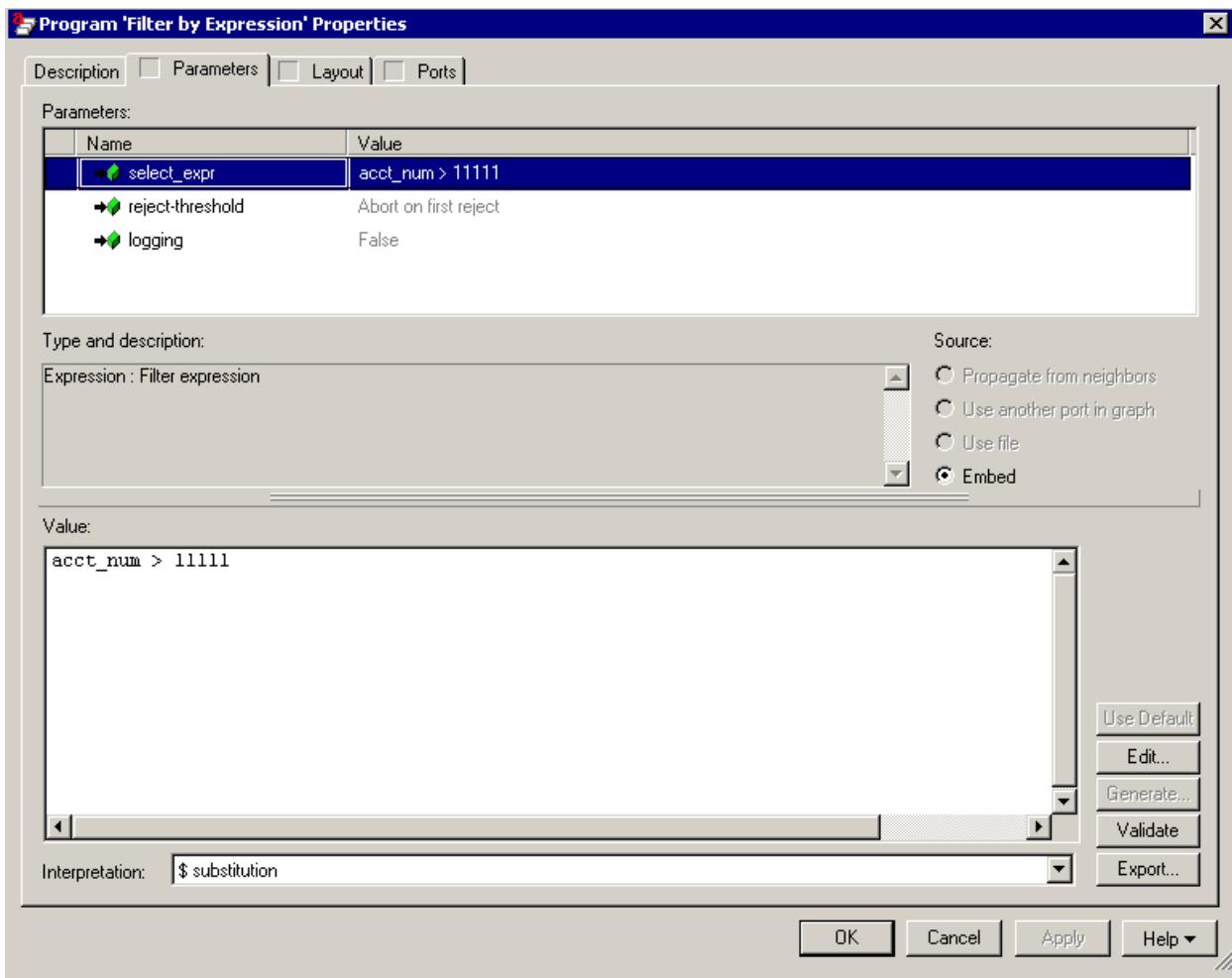
SESSION 3 COMPONENTS

- Filter By Expression
- Reformat
- Dedup Sort
- Join
- Replicate
- Sort
- Meta pivot
- Redefine
- Multistage Transform Components

SAMPLE GRAPH



FILTER BY EXP PROPERTIES



INPUT FILE

View Data: Input File

File Edit View Help

More Records: 100 Go Clear Display

	first_name	last_name	acct_num	balance_amt	
1	aaaa	bbbb	11111	200	
2	cccc	ddd	22222	2000	
3	eee	fffff	33333	30000	
4	aaaa	bbbb	11111	300	
5	eee	fffff	33333	40000	
6	aaaa	bbbb	11111	500	
7	dddd	xyzw	44444	6000	

[EOF]

Scanned 7 records. Retrieved 7 matching selection. (EOF)

FILTER BY EXP O/P

View Data: Fbe_out

File Edit View Help

More Records: 100 Go Clear Display

	first_n...	last_n...	acct_...	balan...
1	ccccc	ddd	22222	2000
2	eee	fffff	33333	40000
3	eee	fffff	33333	30000
4	ddddd	xyzw	44444	6000
[EOF]				

Scanned 4 records. Retrieved 4 matching selection. (EOF)

FILTER BY EXP DESELECT

View Data: Fbe_deselect

File Edit View Help

More Records: 100 Go Clear Display

	first...	last...	acct...	bal...
1	aaaa	bbbb	11111	200
2	aaaa	bbbb	11111	500
3	aaaa	bbbb	11111	300
[EOF]				

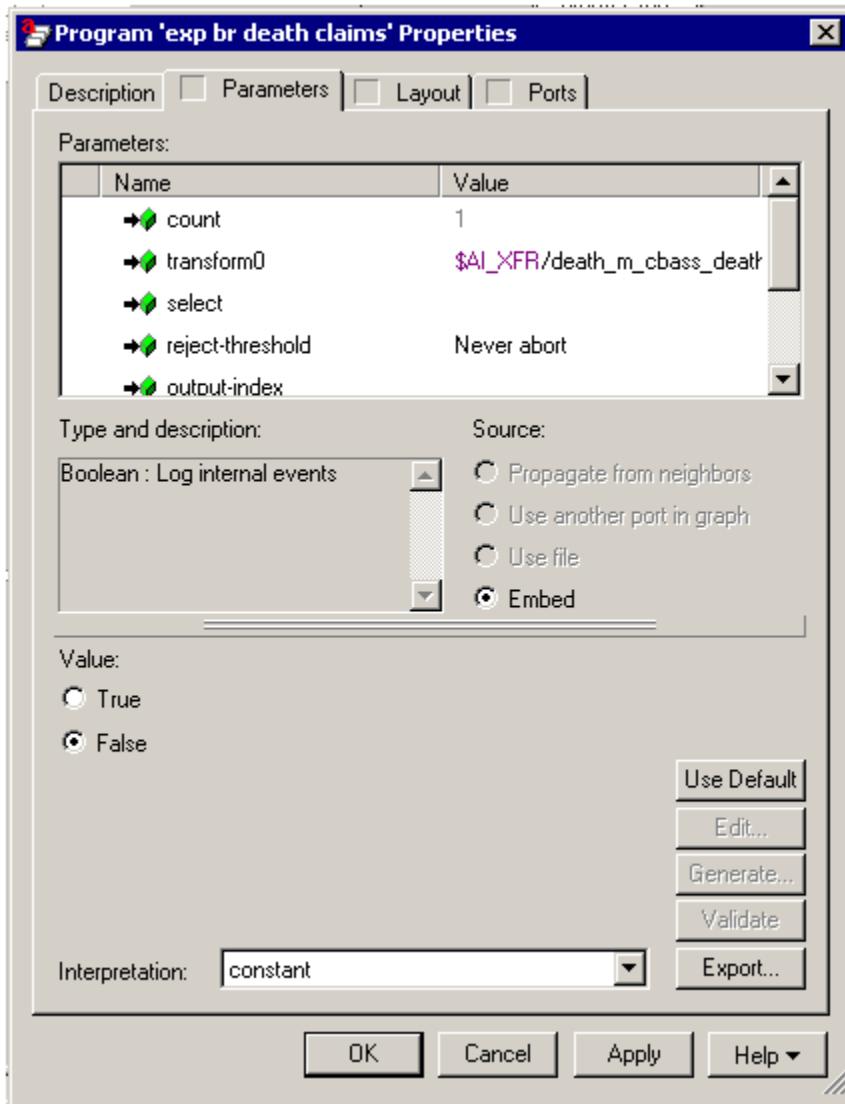
Scanned 3 records. Retrieved 3 matching selection. (EOF)

REFORMAT COMPONENT

Reads records from input port, reformats each according to a transform function (optional in the case of the Reformat Component), and writes the result records to the output (out0) port.

Additional output ports (out1, ...) can be created by adjusting the count parameter.

PARAMETERS



TRANSFORM FUNCTION

- A transform function specifies the business rules used to create the output record.
- Each field of the output record must successfully be assigned a value.

The Transform Editor is used to create a transform function in a graphical manner

REFORMAT PARAMETERS

Parameters of Reformat Component

- **Count**

- **Transform (Xfr) Function**

- **Reject-Threshold**

- ✧ **Abort**

- ✧ **Never Abort**

- ✧ **Use Limit & Ramp**

- ⇒ **Limit**

- ⇒ **Ramp**

INSTRUMENTATION PARAMETERS

- **Limit**

- ✧ **Number of errors to tolerate**

- **Ramp**

- ✧ **Scale of errors to tolerate per input**

- **Tolerance value=limit + ramp*total number of records read**

- **Typical Limit and Ramp settings . .**

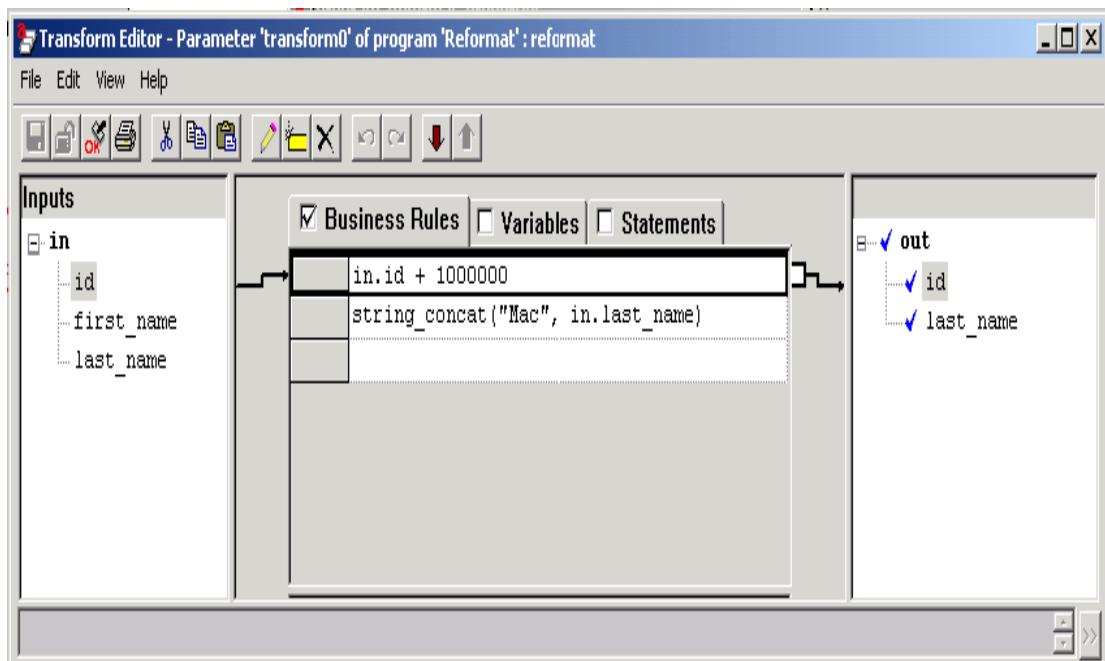
- ✧ **Limit = 0 Ramp = 0.0 □ Abort on any error**

- ✧ **Limit = 50 Ramp = 0.0 □ Abort after 50 errors**

- ✧ **Limit = 1 Ramp = 0.01 □ Abort if more than 1 in 100 records causes error**

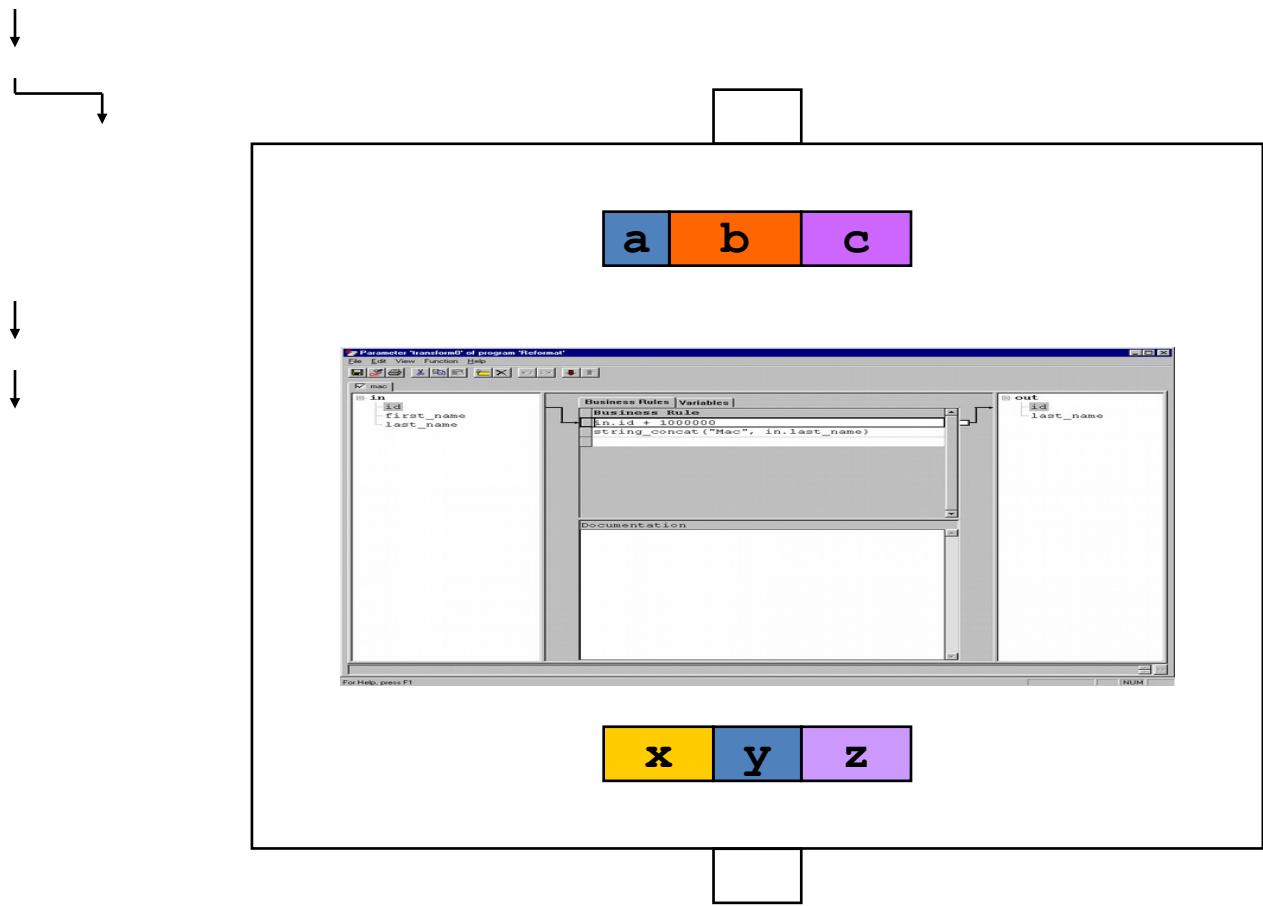
- ✧ **Limit = 1 Ramp = 1 □ Never Abort**

TRANSFORM FUNCTION EDITOR



REFORMAT FUNCTION IN TEXT FORMAT

```
out :: reformat (in) =
begin
  out.id :: in.id + 1000000;
  out.last_name :: string_concat("Mac", in.last_name);
end;
```



A RECORD ARRIVES AT INPUT PORT

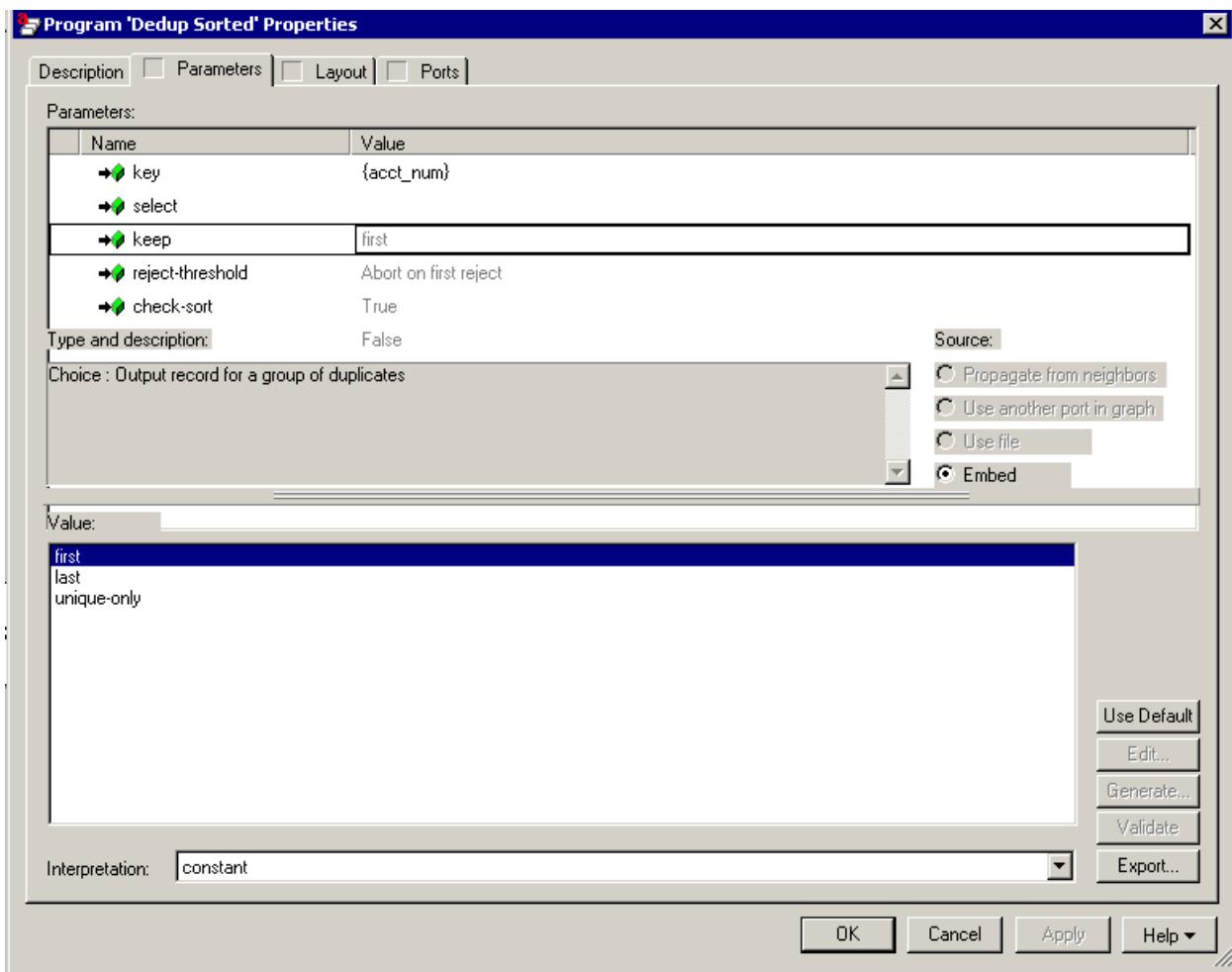
RECORD IS READ INTO COMPONENT

TRANSFORM FUNCTION IS EVALUATED

RECORD WRITTEN TO O/P PORT

DEDUP SORTED

- Separates one specified record in each group of records from the rest of the records in the group
- Requires sorted input



I/P FILES

View Data: Input File

File Edit View Help

More Records: 100 Go Clear Display

	first_name	last_name	acct_num	balance_amt	
1	aaaa	bbbb	11111	200	
2	cccc	ddd	22222	2000	
3	eee	fffff	33333	30000	
4	aaaa	bbbb	11111	300	
5	eee	fffff	33333	40000	
6	aaaa	bbbb	11111	500	
7	dddd	xyzw	44444	6000	

[EOF]

Scanned 7 records. Retrieved 7 matching selection. (EOF)

View Data: Input File

File Edit View Help

More Records: 100 Go Clear Display

	first_name	last_name	acct_num	balance_amt	
1	aaaa	bbbb	11111	200	
2	cccc	ddd	22222	2000	
3	eee	fffff	33333	30000	
4	aaaa	bbbb	11111	300	
5	eee	fffff	33333	40000	
6	aaaa	bbbb	11111	500	
7	dddd	xyzw	44444	6000	

[EOF]

Scanned 7 records. Retrieved 7 matching selection. (EOF)

View Data: Dedup_out

File Edit View Help

More Records: 100 Go Clear Display

	first_name	last_name	acct_num	balance_amt
1	aaaa	bbbb	11111	200
2	cccc	ddd	22222	2000
3	eee	fffff	33333	40000
4	dddd	xyzw	44444	6000
	[EOF]			

Scanned 4 records. Retrieved 4 matching selection. (EOF)

DEDUP DUP

View Data: Dedup_dup

File Edit View Help

More Records: 100 Go Clear Display

	first...	last_n...	acct_...	balan...	
1	aaaa	bbbb	11111	500	
2	aaaa	bbbb	11111	300	
3	eee	fffff	33333	30000	
[EOF]					

Scanned 3 records. Retrieved 3 matching selection. [EOF]

JOIN COMPONENT

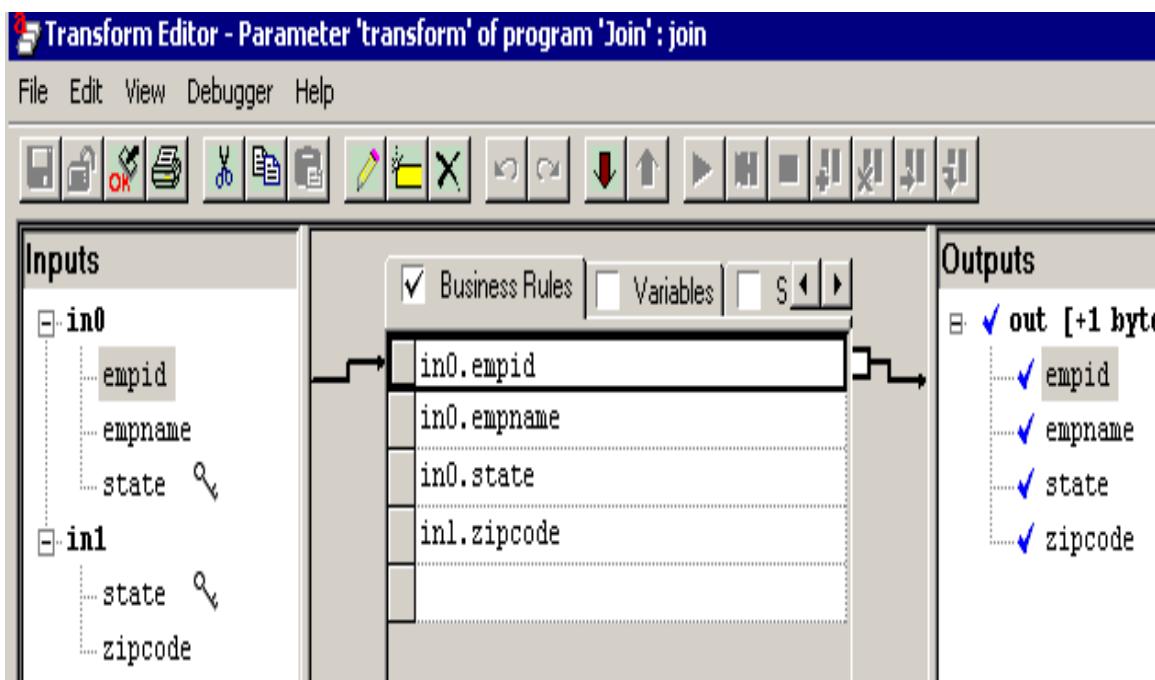
Join performs a join of inputs. By default, the inputs to join must be sorted and an inner join is computed.

Note: The following slides and the on-line example assume the join-type parameter is set to 'Outer', and thus compute an outer join.

Reads records from multiple input ports

Operates on records with matching keys using a multi-input transform function

Writes result to the output port



LOOK INSIDE JOIN

***join-type = Full Outer join**

RECORDS ARRIVE IN JOIN

RECORDS READ INTO JOIN COMPONENT

INPUT FIELDS ARE COMPARED

ALIGNED RECORDS PASS THROUGH TRANSFORM FUNCTION

TRANSFORM ENGINE EVALUATE BASED ON INPUT

A result record is emitted and written out as long as all output fields have been successfully computed

A result record is emitted and written out as long as all output fields have been successfully computed

New records arrive at the inputs of the Join

Again, they are read into the Join component

Again, they are read into the Join component

INPUT KEY FIELDS ARE COMPARED

The aligned records are passed to the transformation function

The transformation engine evaluates based on the inputs

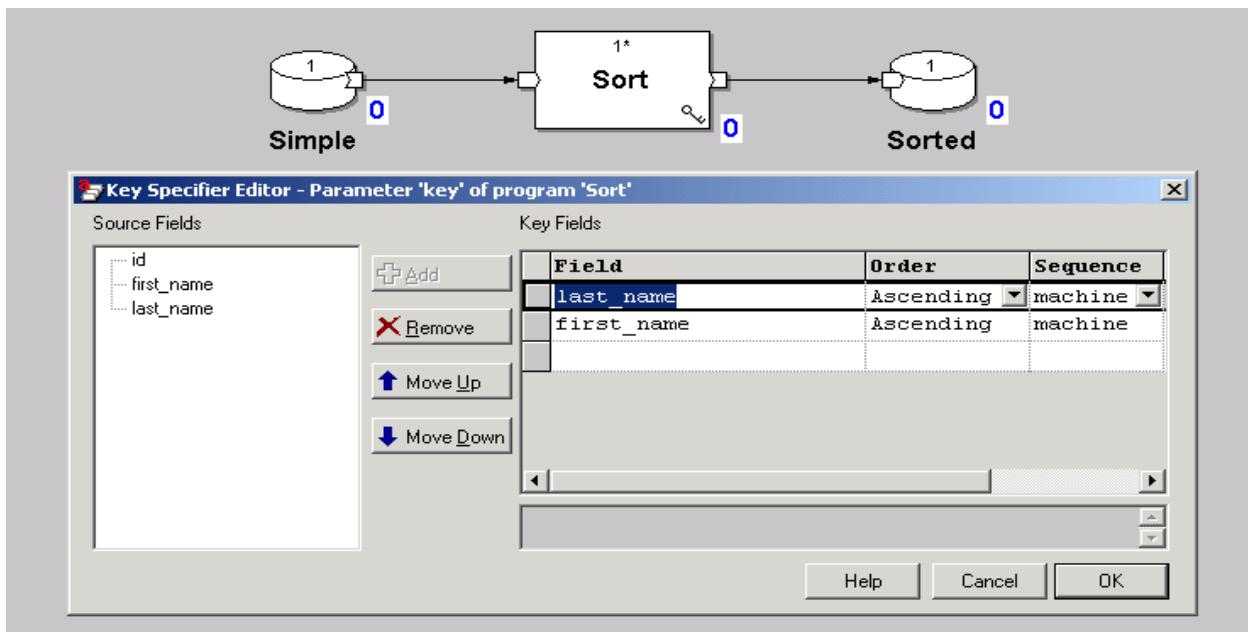
A result record is generated and written out

REPLICATE

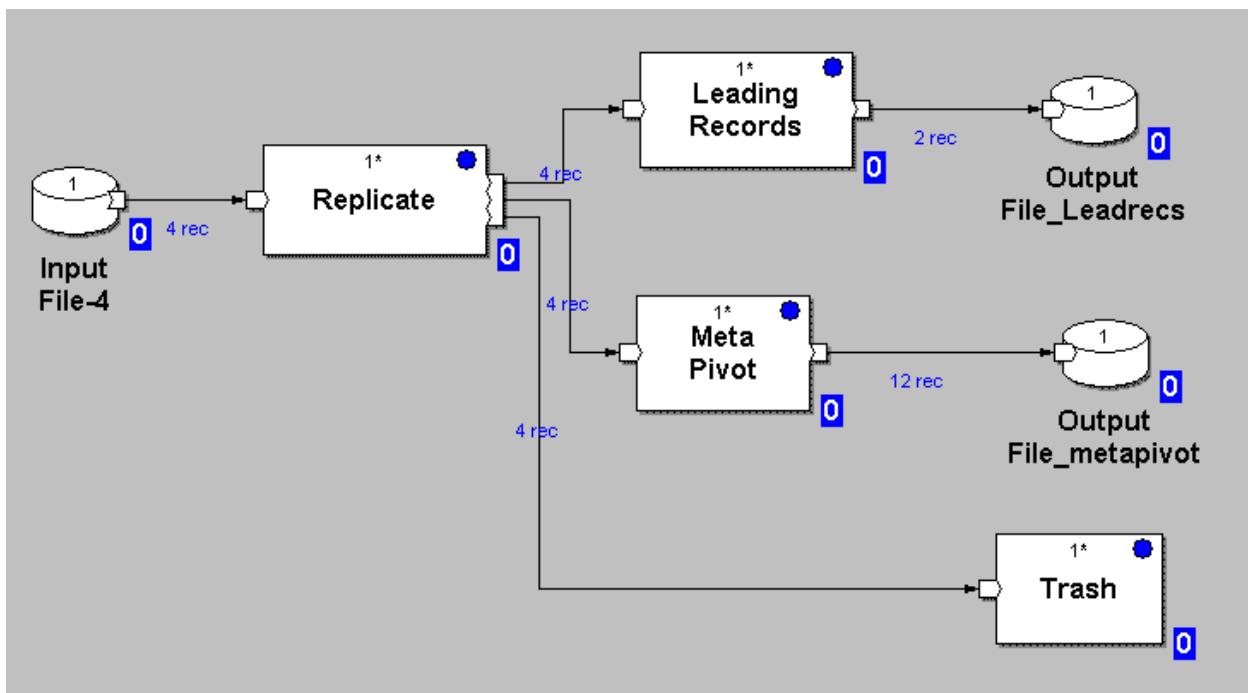
- I Arbitrarily combines the records from all the flows on the in port into a single flow**
- I Copies that flow to all the flows connected to the out port**

Sort Component :_

- I Reads records from input port, sorts them by key, writes result to output port**
- I Parameters**
 - ◊ Key**
 - ◊ Max-core**



LEADING RECORDS META PIVOT TRASH



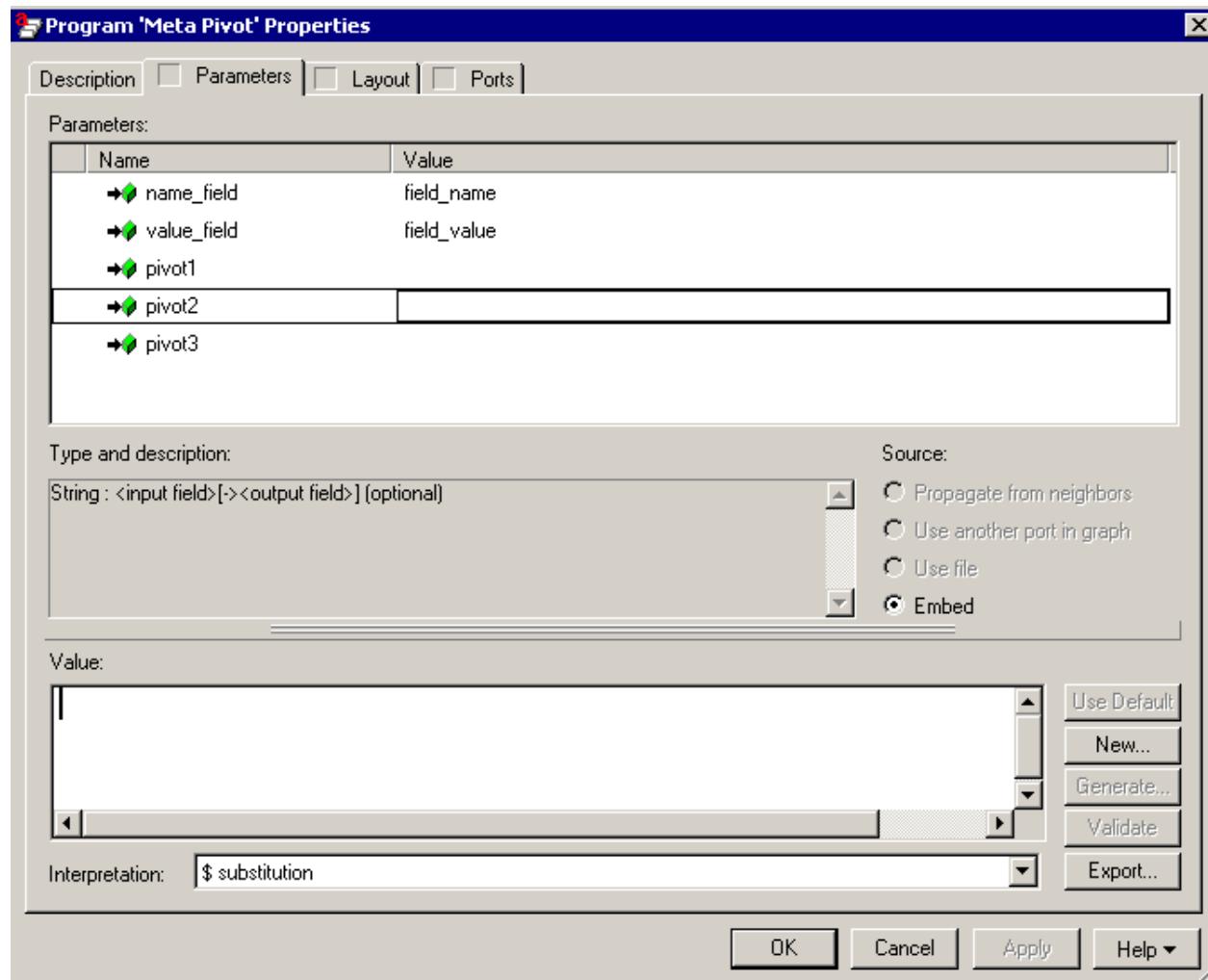
LEADING RECORD

- **Leading Records** copies a specified number of records from its in to its out port, counting from the first record in the input file.
- **Parameter**

num_records

(integer,required)

Specifies the number of records to copy from in port to out port.



□ Purpose

- **Meta Pivot splits records by data fields (columns), converting each input record into a series of separate output records. There is one separate output record for each field of data in the original input record. Each output record contains the name and value of a single data field from the original input record.**

- Consider a simple graph with an input file of product data. Each record consists of three fields (columns) of data. Its record format is as follows:
- record
- **string(7) product_cd = "";**
- **string(,") product_name = "";**
- **decimal(9.2) whs_price = 0.0;**
- **string("\n") newline;**
- **end;**
- A Meta Pivot component reads this input file and then writes the results into an output file:

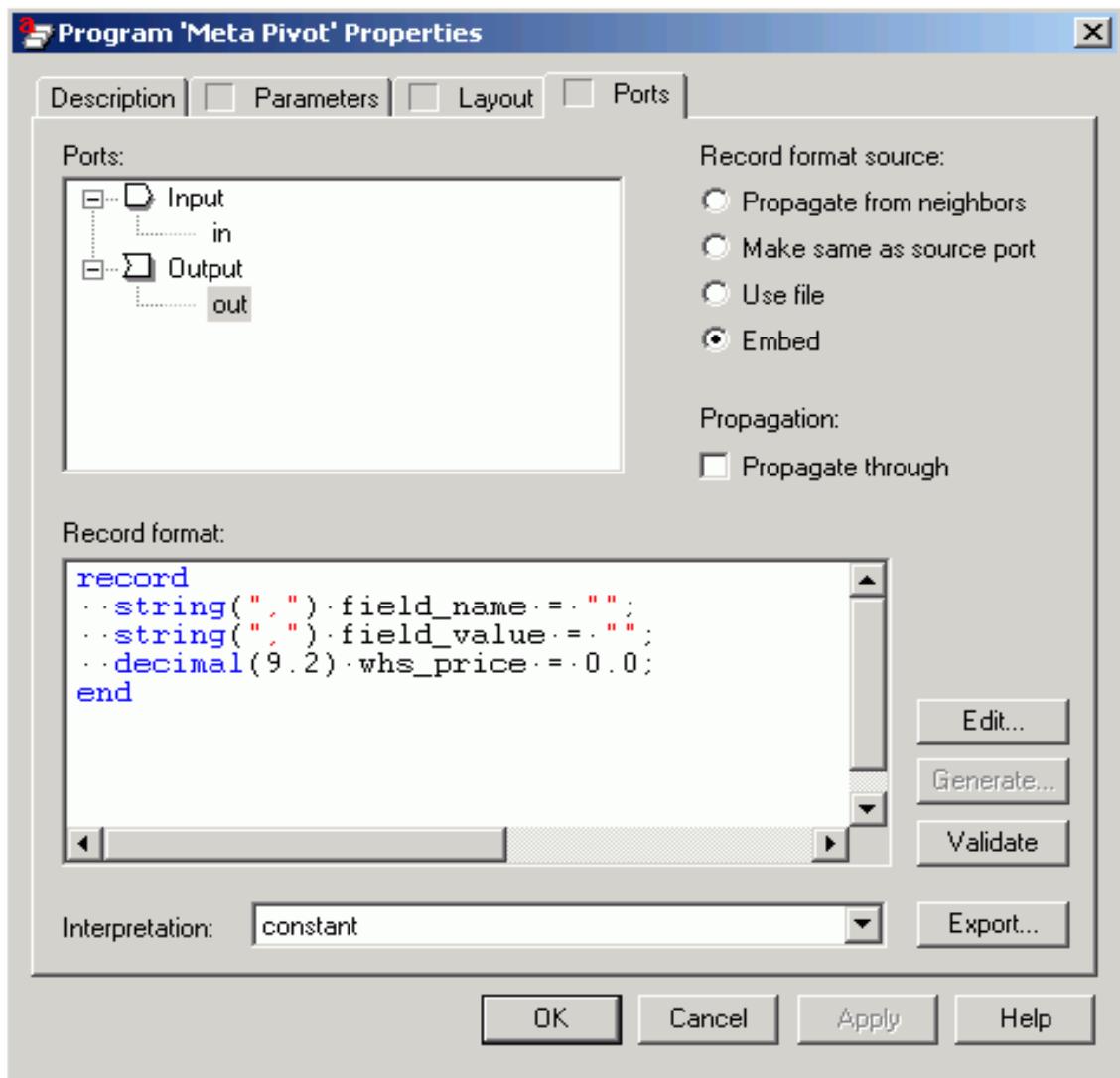
View Data: Input File

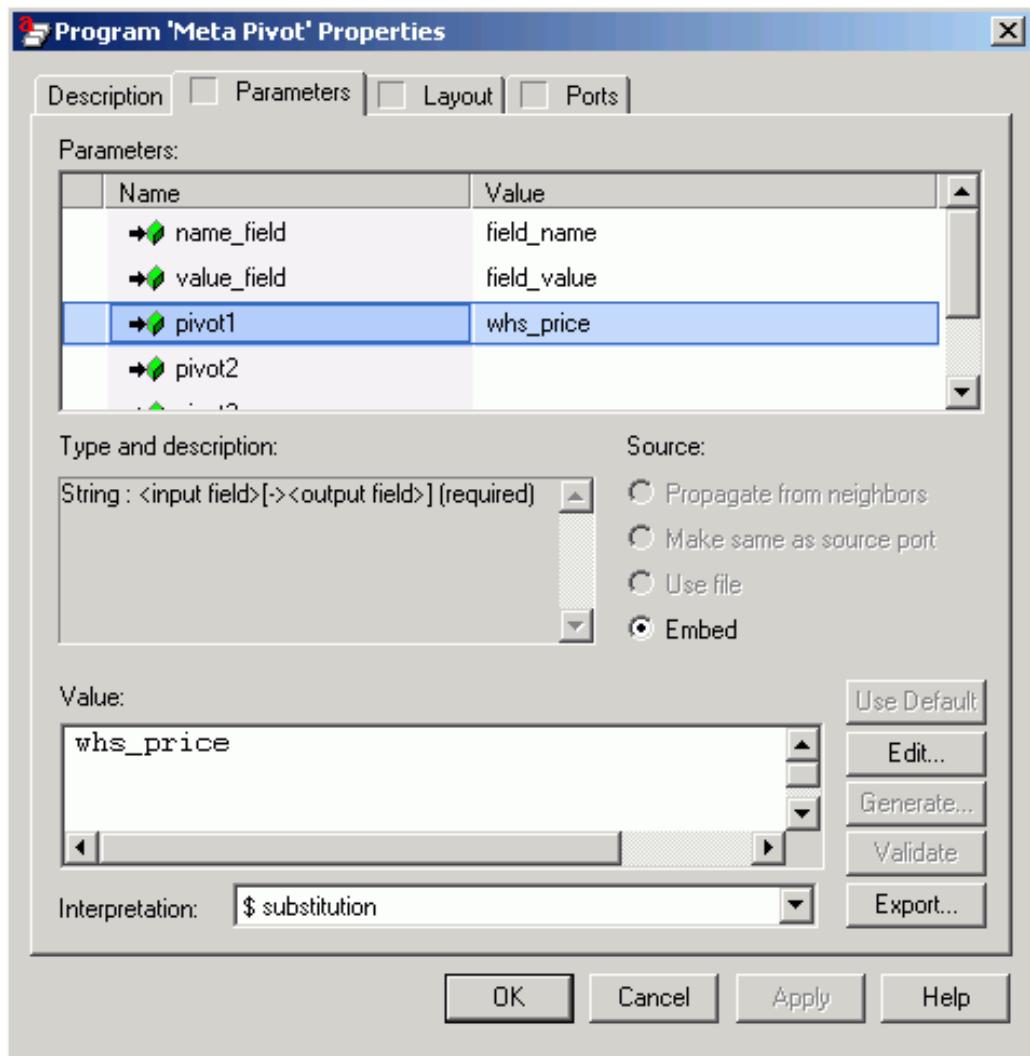
File Edit View Help

More Records: 100 Go Clear Display

	product_cd	product_name	whs_price	newl...
1	ANQ1758	15/16 in. Screwdriver	8.07	\r\n
2	ASH3788	11/16 in. Wrench	6.01	\r\n
3	BAI4650	7/8 in. Drywall	2.41	\r\n
4	BAW9213	3/4 in. Screwdriver	2.96	\r\n
5	BCP3953	5/8 in. Drillbit	0.74	\r\n
6	BGK9924	1/4 in. Screw	0.38	\r\n
7	BHG6488	1/2 in. Bolt	0.26	\r\n
8	BQR2724	1/16 in. Padding	0.52	\r\n
9	BXY1598	5/8 in. Padding	0.99	\r\n
10	CCN2562	7/16 in. Wrench	5.73	\r\n
	[EOF]			

Scanned 10 records. Retrieved 10 matching selection. (EOF)





View Data: Output File

File Edit View Help

More Records: 100 Go Clear Display

	field_name	field_value	whs_price	
1	product_cd	ANQ1758	8.07	
2	product_name	15/16 in. Screwdriver	8.07	
3	newline	\r\n	8.07	
4	product_cd	ASH3788	6.01	
5	product_name	11/16 in. Wrench	6.01	
6	newline	\r\n	6.01	
7	product_cd	BAI4650	2.41	
8	product_name	7/8 in. Drywall	2.41	
9	newline	\r\n	2.41	
10	product_cd	BAW9213	2.96	
11	product_name	3/4 in. Screwdriver	2.96	
12	newline	\r\n	2.96	
13	product_cd	BCP3953	0.74	
14	product_name	5/8 in. Drillbit	0.74	
15	newline	\r\n	0.74	
16	product_cd	BGR9924	0.38	
17	product_name	1/4 in. Screw	0.38	
18	newline	\r\n	0.38	
19	product_cd	BRSC400	0.26	

Scanned 30 records. Retrieved 30 matching selection. (EOF)

TRASH

Purpose

Trash ends a flow by accepting all records in it and discarding them.

TRASH is a broadcast component without an out port.

Parameters

None

Runtime behavior

Trash does the following:

1. Reads records from the in port
2. Discards the records.

REDEFINE FORMAT

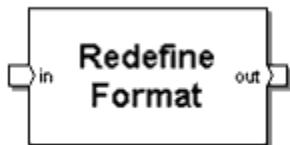
Similar to reformat.

Purpose

Redefine Format copies records from its in port to its out port without changing the values in the records.

It doesn't have transform function.

You can use Redefine Format to improve graph performance when you reduce the number of fields in an input record — by renaming the fields without changing the values in the records



Multistage Transform Components

- I Data transformation in multiple stages following several sets of rules**
- I Each set of rule form one transform function**
- I Information is passed across stages by temporary variables**
- I Stages include initialization, iteration, finalization and more**
- I Few multistage components are aggregate, rollup, scan**

Rollup

Aggregate

Scan

Generates summary records for group of input records

ROLLUP

What it does?

Rollup evaluates a group of input records that have the same key, and then generates records that either summarize each group or select certain information from each group.

Aggregate functions:

□ **Count**

□ **Sum**

□ **Min**

□ **Max**

□ **Avg**

□ Product

□ First

□ Last

□ stdev

□

Parameters

sorted-input

(boolean, required)

➤ In memory: Input need not be sorted

➤ Input must be sorted or grouped

- Default is Input must be sorted or grouped.

key-method

(choice, optional)

⇒ Use key specifier

⇒ Use key_change function

key

(key specifier, optional)

transform

(filename or string, required)

Problem:

Given the List of customer ids and their transaction dates and amounts. Find the day when the customer begins his transaction and the day of his latest transaction with the total amount for which he has purchased.

ROLLUP EXAMPLE

The screenshot shows a software application window titled "View Data: Input File". The window has a menu bar with "File", "Edit", "View", and "Help". Below the menu is a toolbar with various icons. A search bar displays "More Records: 100" and a "Go" button. There is also a "Clear Display" checkbox. The main area is a table with the following columns: dt, cust_id, amount, and newline. The data consists of 10 rows, each representing a transaction. Row 10 is labeled "[EOF]". At the bottom of the window, there is a status bar with the text "Export selection to a file" and a help icon.

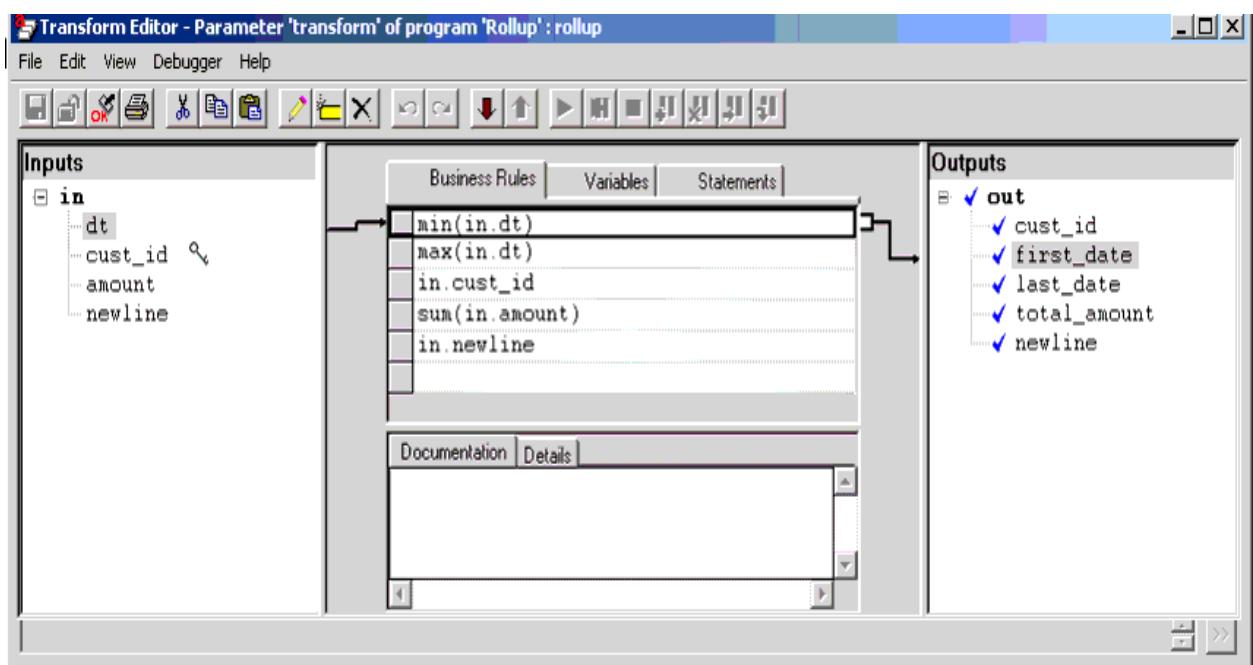
	dt	cust_id	amount	newline
1	1996.09.27	00008372	024550	\r\n
2	1996.09.19	00008372	043595	\r\n
3	1996.09.02	00008225	009915	\r\n
4	1996.09.19	00007888	015905	\r\n
5	1996.09.30	00008225	009915	\r\n
6	1996.09.23	00008372	034500	\r\n
7	1996.09.26	00007888	014500	\r\n
8	1996.09.15	00008225	013245	\r\n
9	1996.09.25	00007888	032415	\r\n
10	1996.09.21	00007888	041590	\r\n
	[EOF]			

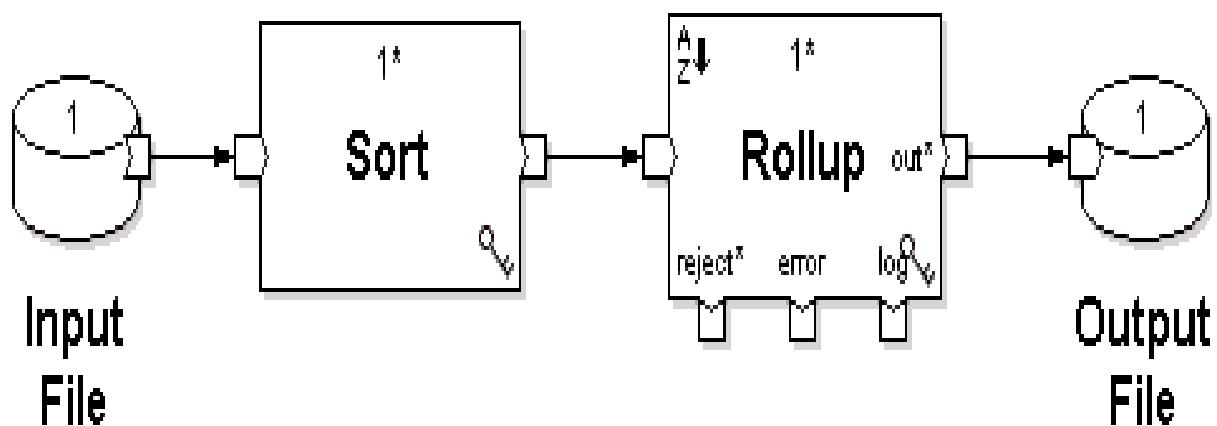
Record Format Editor - port 'read' of file 'Input File'

File Edit View Help

OK

```
01 record
02   ··date("YYYY.MM.DD")·dt;
03   ··string(8)·cust_id;
04   ··decimal(6)·amount;
05   ··string(2)·newline;
06 end
```





Record Format Editor - port 'write' of file 'Output File'

File Edit View Help

OK

```

01 record
02   .string(8) · cust_id;
03   .date("YYYY.MM.DD") · first_date;
04   .date("YYYY.MM.DD") · last_date;
05   .decimal(6) · total_amount;
06   .string(2) · newline;
07 end
  
```

	cust_id	first_date	last_date	total_amount	newline
1	00007888	1996.09.19	1996.09.26	104410	\r\n
2	00008225	1996.09.02	1996.09.30	33075	\r\n
3	00008372	1996.09.19	1996.09.27	102645	\r\n

SCAN

What it does?

For every input record, Scan generates an output record that includes a running cumulative summary for the group the input record belongs to. For example, the output records might include successive year-to-date totals for groups of records.

Parameters

sorted-input

(boolean, required)

- In memory: Input need not be sorted
 - Input must be sorted or grouped
- ◊ Default is Input must be sorted or grouped.

key-method

(choice, optional)

- ⇒ Use key specifier
- ⇒ Use key_change function

key

(key specifier, optional)

transform

(filename or string, required)

SCAN – TRANSFORM FUNCTIONS

- Input select
- Initialize
- Scan
- Finalize
- output select
- key change

EXAMPLE

customer_id	dt	amount
C002142	1994.03.23	52.20
C002142	1994.06.22	22.25
C003213	1993.02.12	47.95
C003213	1994.11.05	221.24
C003213	1995.12.11	17.42
C004221	1994.08.15	25.25
C008231	1993.10.22	122.00
C008231	1995.12.10	52.10

EXAMPLE

```
type temporary_type =
  record
    decimal(8.2) amount_to_date;
  end;

temp :: initialize(in) =
begin
  temp.amount_to_date :: 0;
end;

out :: scan(temp, in) =
begin
  out.amount_to_date ::

    temp.amount_to_date + in.amount;
end;

out :: finalize(temp, in) =
begin
  out.customer_id   :: in.customer_id;
  out.dt          :: in.dt;
  out.amount_to_date :: temp.amount_to_date;
end;
```

customer_id	dt	amount_to_date
C002142	1994.03.23	52.20
C002142	1994.06.22	74.45
C003213	1993.02.12	47.95
C003213	1994.11.05	269.19
C003213	1995.12.11	286.61
C004221	1994.08.15	25.25
C008231	1993.10.22	122.00
C008231	1995.12.10	174.10

NORMALIZE

Why?

Normalize generates multiple output records from each of its input records. You can directly specify the number of output records for each input record, or the number of output records can depend on some calculation.

TRANSFORM FUNCTIONS

Transform functions:

- **input_select**
- **initialize**
- **length**
- **Normalize**
- **finalize**
- **output select**

View Data: Input File

File Edit View Help

More Records: 100 Go Clear Display

	customer_id	ntrans	transactions	newline
			transdate	amount
1	c002142	02	[vector] [0] 1994.03.23 [1] 1994.06.22	\r\n00052.20 00022.25
2	c003213	03	[vector] [0] 1993.02.12 [1] 1994.11.05 [2] 1995.12.11	\r\n00047.95 00221.24 00017.42
3	c004221	01	[vector] [0] 1994.08.15	\r\n00025.25
4	c008231	02	[vector] [0] 1993.10.22 [1] 1995.12.10	\r\n00122.00 00052.10
	[EOF]			

Scanned 4 records. Retrieved 4 matching selection. (EOF)

Record Format Editor - port 'read' of file 'Input File'

File Edit View Help

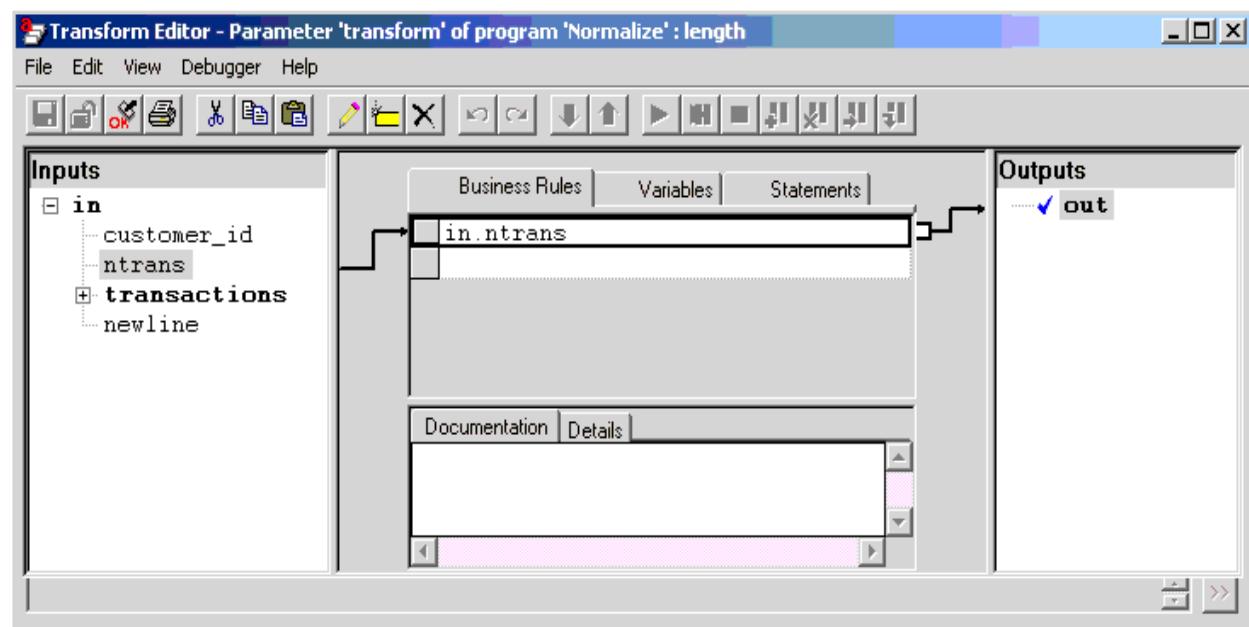
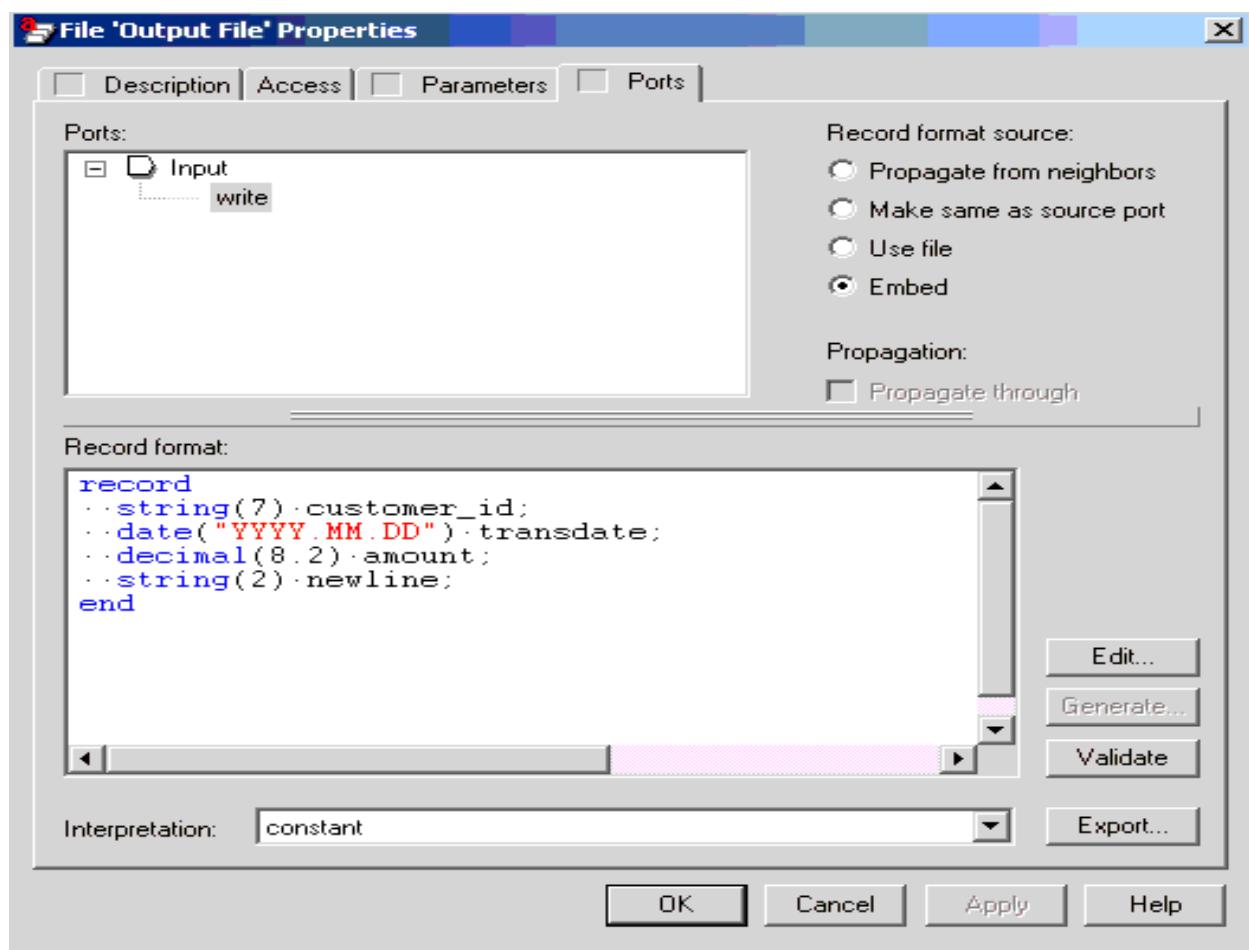
OK

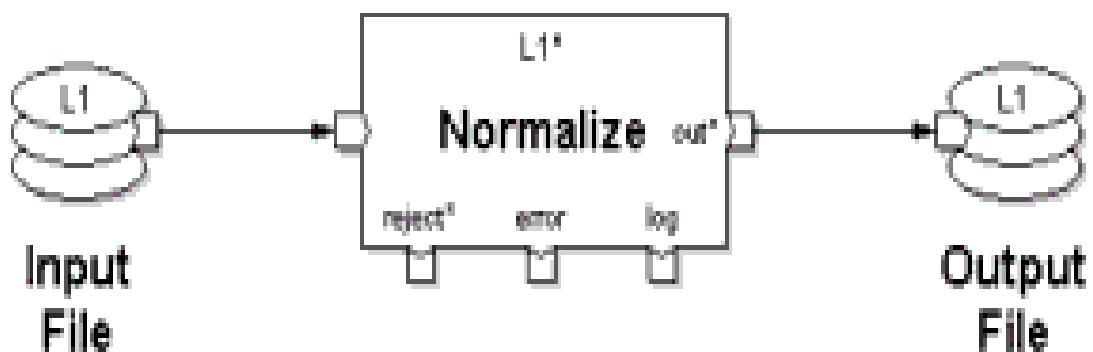
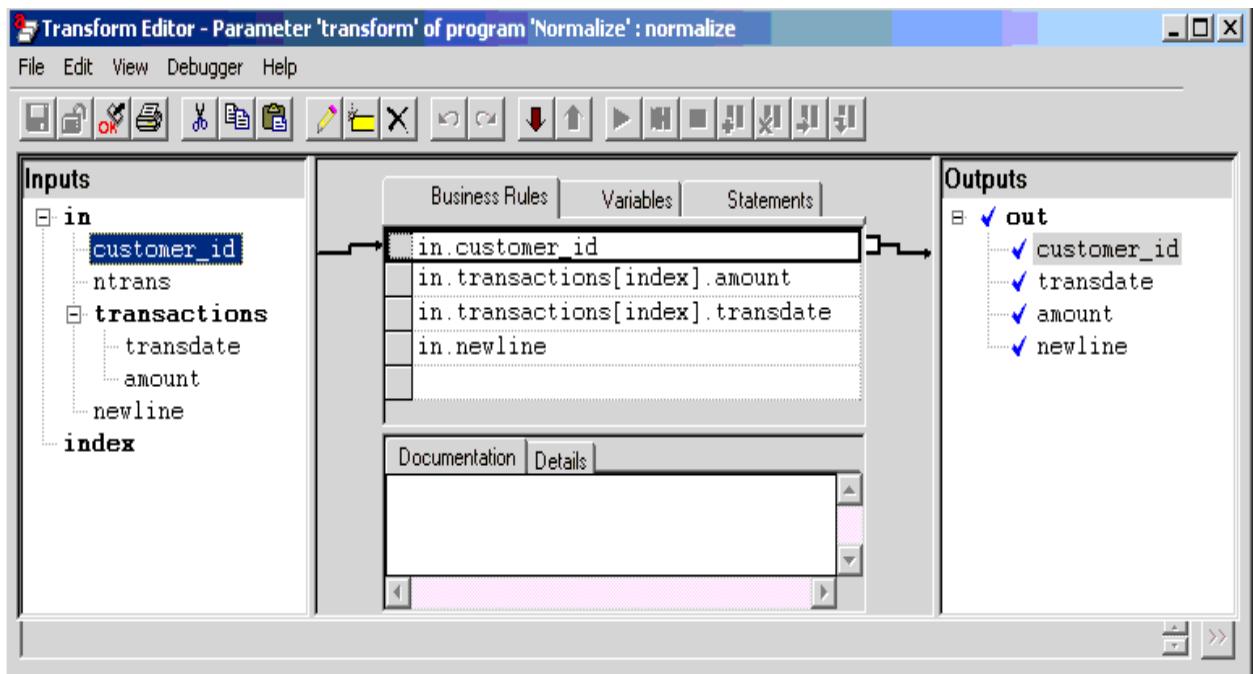
```

01 record
02   ·string(7)·customer_id;
03   ·decimal(2)·ntrans;
04   ·record
05     ···date("YYYY.MM.DD")·transdate;
06     ···decimal(8.2)·amount;
07   ··end·transactions[ntrans];
08   ·string(2)·newline;
09 end

```

EXAMPLE





View Data: Output File

File Edit View Help

More Records: 100 Go Clear Display

	customer_id	transdate	amount	newline
1	C002142	1994.03.23	00052.20	\r\n
2	C002142	1994.06.22	00022.25	\r\n
3	C003213	1993.02.12	00047.95	\r\n
4	C003213	1994.11.05	00221.24	\r\n
5	C003213	1995.12.11	00017.42	\r\n
6	C004221	1994.08.15	00025.25	\r\n
7	C008231	1993.10.22	00122.00	\r\n
8	C008231	1995.12.10	00052.10	\r\n
	[EOF]			

DENORMALIZE

Denormalize Sorted consolidates groups of related records by key into a single output record with a vector field for each group, and optionally computes summary fields in the output record for each group.

- **input_select**
- **temporary_type**
- **denormalize**
- **finalize**
- **output_select**
- **key_change**

View Data: Input File

File Edit View Help

More Records: 100

	customer_id	date_time	amount	newline	
1	C002142	2001.04.20	00055.63	\n	
2	C002142	2004.01.15	00010.11	\n	
3	C002142	2004.02.27	00079.80	\n	
4	C002142	2004.03.17	00452.93	\n	
5	C002142	2004.04.10	01052.27	\n	
6	C002142	2004.05.10	00000.12	\n	
7	C002142	2004.06.06	00069.20	\n	
8	C002142	2005.06.22	00022.25	\n	
9	C002142	2006.01.27	00052.20	\n	
10	C002313	2003.11.05	00221.24	\n	
11	C002313	2004.08.07	00111.20	\n	
12	C002313	2004.09.15	00097.17	\n	
13	C002313	2005.11.30	00061.05	\n	
14	C003213	2001.02.12	00047.95	\n	
15	C003213	2004.12.11	00017.42	\n	
16	C004221	2002.08.15	00025.25	\n	
17	C008231	2001.10.22	00122.00	\n	
18	C008231	2003.01.30	00043.00	\n	
19	C008231	2005.12.10	00052.10	\n	
	[EOF]				

```

type element_type =
decimal(8.2);

type denormalization_type =
element_type[20];

type temporary_type =
record
    decimal(8.2) total_amount;
    decimal(4) count;
end;

out::initial_denormalization() =

```

```
begin
out :: 0;
end;

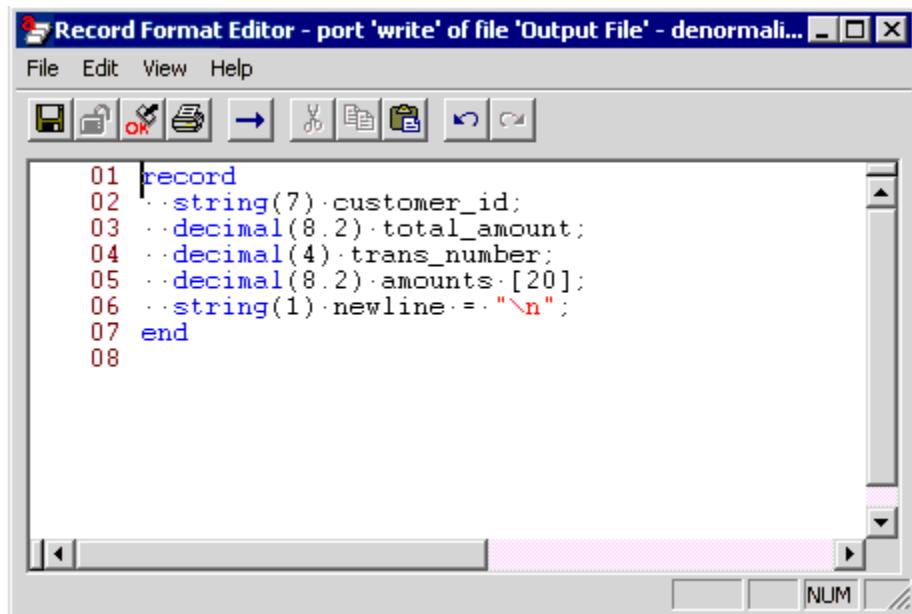
/* Initialize the rollup's temporary storage... */
temp::initialize(in) =
begin
temp.total_amount :: 0;
temp.count :: 0;
end;

/* Rollup on normalize... */
temp::rollup(temp, in) =
begin
temp.total_amount :: temp.total_amount + in.amount;
temp.count :: temp.count + 1;
end;

/* Do computation... */
denorm_out::denormalize(temp, denorm, in, count) =
begin
denorm_out.index :: count;
denorm_out_elt :: in.amount;
denorm_out.update :: count < 100;
end;

/* Create output record */
out::finalize(temp, denorm, in) =
begin
```

```
out.trans_number :: temp.count;  
out.total_amount :: temp.total_amount;  
out.customer_id :: in.customer_id;  
out.amounts :: denorm;  
end;
```



View Data: Output File

	customer_id	total_amount	trans_number	amounts	newline	
1	C002142	• 1794.51	••• 9	[vector]	\n	
2	C002313	•• 490.66	••• 4	[vector]	\n	
3	C003213	••• 65.37	••• 2	[vector]	\n	
4	C004221	••• 25.25	••• 1	[vector]	\n	
5	C008231	•• 217.10	••• 3	[vector]	\n	
	[EOF]					

FUNCTIONS

- **Lookup Function:**
- **Returns a data record from a lookup file. The syntax is record lookup (string file_label, [expression]).**
- **If a record matches the values of the expression arguments, the function returns the matching record. (This is true even if one or more expressions are NULL.)**
- **If no record matches the values of the expression arguments, the function returns NULL .**
- **If there are multiple matching records or if there are no expression arguments, the function returns an arbitrary record.**

Example:

- **Suppose a graph has a lookup file named “structure_code.dat” with fields structure_code, structure_name and level where the key specifier is {structure_code}. The lookup function can be used is: Lookup(“structure_code.dat”,in.structure_code) to find out the record which matches the structure_code.**

- If I need only structure_name instead of the full record the the expression can be
`Lookup("structure_code.dat",in.structure_code).structure_name.`
- NB: **Lookup_local:** Returns a data record from a partition of a lookup file. We can use this function in case the file is partitioned.

OTHER LOOKUP FUNCTIONS

- **lookup_count :** Returns the number of matching data records in a lookup file.
- **lookup_match:** Looks for a specific match from the lookup file. It returns 1 if there is any record, NULL if there is no record matching, 1 if more than one record, 0 if the file is empty.
- **lookup_nth:** Returns a specific record from a lookup file. The syntax is record `lookup_nth (string file_label, unsigned integer record_num)`

STRING FUNCTIONS

- **string_substring :**This function returns a substring of a string; the substring can be a specific length, and begin at a specific character
Example: `string_substring("abcdefg", 4, 3) -> "def"`
- **string_index :** This function returns the index of the first character of the first occurrence of a string within another string.
Example: `string_index("nomination","i") -> 4`
- **string_ltrim :** This function returns a string trimmed of leading and trailing blank characters.
Example: `string_ltrim(" abc ") -> "abc"`
- **String_trim:** This function returns a string trimmed of trailing blank characters.
Example: `string_trim("abc ") -> "abc"`
- **string_filter :** This function returns the characters of one string which also appear in another.
`string_filter("AXBYCZ", "ABCDEF") -> "ABC"`

- **string_filter_out** : This function returns the characters of one string which do not appear in another.

Example: `string_filter_out("AXBYCZ", "ABCDEF") ->"XYZ"`

- **is_blank** : This function tests whether a string contains only blank characters.

Example: `is_blank("")-> 1,`

`is_blank("abc") -> 0`

- **is_defined** : This function tests whether an expression is not NULL.

Example: `is_defined(123) -> 1`

- **is_null** : This function tests whether an expression is NULL.

Example: `is_null(123) -> 0`

- **is_error** : This function tests whether an error occurs while evaluating an expression.

Example: `is_error(a + b) -> 0`

DATE FUNCTIONS

- **date_add_months** : This function returns the internal representation of a date resulting from adding (or subtracting) a number of months to the specified date.

Example: `(date("YYYY-MM-DD"))date_add_months((date("YYYYMMDD")) "20000229",12) -> "2001-02-28"`

- **date_day** : This function returns the day of the month of a date.

Example: `date_day((date("YYYYMMDD")))"19980518") -> 18`

Like that there are lots of function is being used in ab initio.

- **date_day_of_month**
- **date_day_of_week**
- **date_day_of_year** etc...

MATH FUNCTIONS

There are several math functions available for different calculation like

- **math_abs**
- **math_exp**
- **ceiling**
- **decimal_round :**
- **decimal_round_down**
- **decimal_round_up**
- **decimal_truncate**

ASSIGN KEY COMPONENT

Assign key component is being used to create surrogate key which is the unique key in the table.

It has the following ports:

- **In**
- **Key**
- **Old**
- **New**
- **First**
- **Assign Keys reads input records on the in port and checks them against input records on the key port. For each record on the in port, Assign Keys assigns a value to a surrogate key field. The assigned value is based on the value of the natural key field in the same input record.**

ASSIGN KEY COMPONENT

- **The first output port receives a record for each new surrogate key which can be used to update the information source for the key port**

- **The new output port receives a record for each input record for which a new surrogate key was generated and assigned.**
- **The old output port receives a record for each input record to which an existing surrogate key was assigned.**

VECTOR

- **A vector is an array of same type of elements that is repeated.**
- **Number of repeats is a constant integer or the value of another field in the record.**

Example:

The following example produces a vector of three integers: 1, 2, and 3:

[vector 1, 2, 3]

The following example produces a vector of three strings:

[vector "a", "little", "example"]

The following example produces a vector of two records, each with fields id and name.

[vector

[record

id 123343

name "John Smith"],

[record

id 123344

name "Mary Jones"]]

CUSTOM COMPONENT

- **A custom component, as opposed to a built-in Ab Initio component, is a component that you build from a template.**

1. Creating Custom Components Based on Built-in Components:

To configure an Ab Initio built-in component and then save it for reuse as a custom component:

- **Insert the component you want to use in the graph.**
- **Double-click the component to open its Properties,**
- **Then do whatever change you want to make.**
- **In the Properties dialog, click OK.**
- **Select the component.**
- **On the Menu bar of the GDE, choose File > Save Component "Label" As.**
- **To add the new component to the the Component Organizer, save the component in the installed components directory, typically C:\Program Files\Ab Initio\Ab Initio GDE\Components.**

2. Creating Custom Components from the Component Organizer:

To create custom components from the Component Organizer, follow these steps:

- Select a location in the the Component Organizer for the custom component:
for example, the My Components folder. To add a subfolder, right click My Components and choose New > Folder on the shortcut menu.
- Right click My Components and choose what type of component you want to create.

The label Newtype_of_component of the new component will appear

- In the Component Organizer, right-click the Newtype_of_component label and choose Edit or Edit as text from the shortcut menu to modify the .mpc file or .mdc file.
- In the Component Organizer, click to select the Newtype_of_component , label, then click again and overtype to change the name.

Now the component can be dragged from the Component Organizer onto the graph. A component icon with the same label appears in the graph.

CONDITIONAL COMPONENT

Execution of any component can be restricted based of any condition.

In the Properties>condition tab the condition has to be specified. The execution of that component would be dependent on that.

PARALLELISM

Parallel Runtime Environment

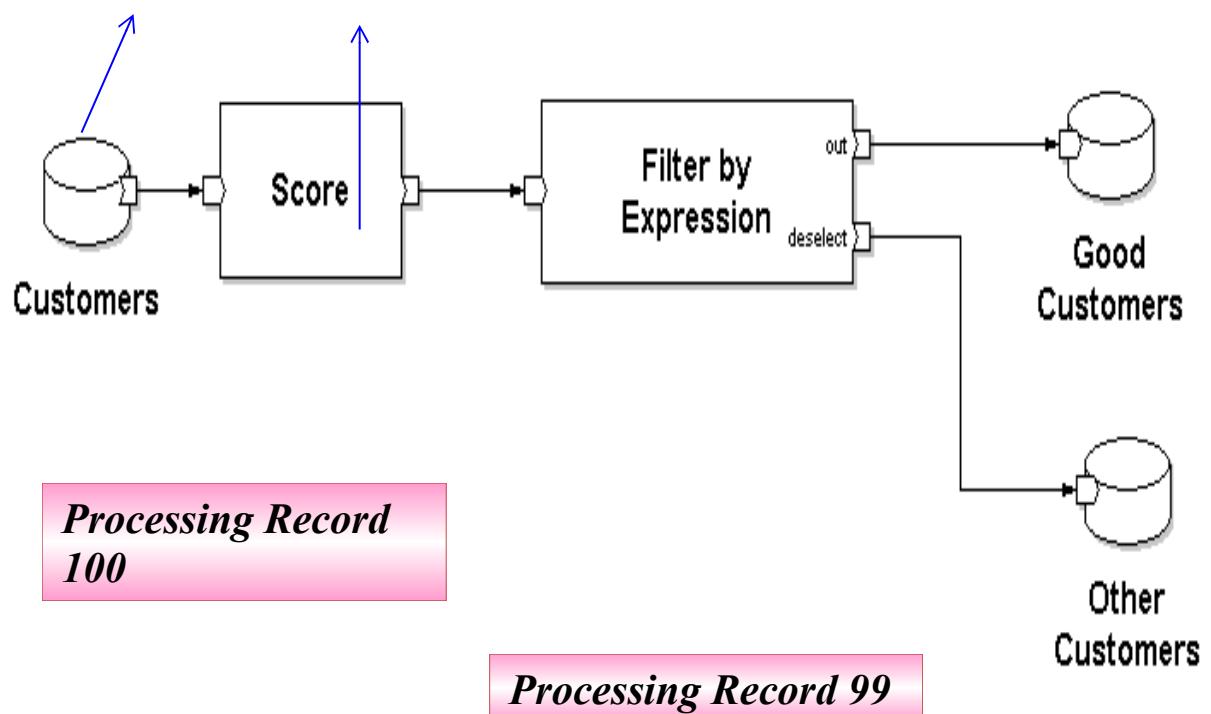
Where some or all of the components of an application – datasets and processing modules are replicated into a number of partitions, each spawning a process.

Ab Initio can process data in parallel runtime environment

Forms of Parallelism

- Component Parallelism
- Pipeline Parallelism
- Data Parallelism

COMPONENT PARALLELISM – DIFFERENT INSTANCES OF SAME COMPONENT RUN ON DIFFERENT DATASETS



NOTE : # of processes per component = # of partitions

Multifiles

A global view of a set of ordinary files called *partitions* usually located on different disks or systems

Ab Initio provides shell level utilities called "*m_commands*" for handling Multifiles (copy, delete, move etc.)

Multifiles reside on Multidirectories

Each is represented using URL notation with
"mfile" as the protocol part:
`//host1/u1/jo/mfs`

mfile://pluto.us.com/usr/ed/mfs1/new.dat

- Data parallelism scales with data and requires data partitioning
- Data can be partitioned using different partitioning methods.
- The actual way of working in a parallel runtime environment is transparent to the application developer.
- It can be decided at runtime whether to work in serial or in parallel, as well as to determine the degree of parallelism

MULTIFILE COMPONENTS

- m_mkfs
- m_mkdir
- m_ls

- m_expand
- m_dump
- m_cp
- m_mv
- m_touch
- m_rm

M_MKFS COMMAND

m_mkfs *mfss-url dir-url1 dir-url2 ...*

- Creates a multifile system rooted at *mfss-url* and having as partitions the new directories *dir-url1*, *dir-url2*, ...

```
$ m_mkfs //host1/u/jo/mfs3 \
    //host1/vol4/dat/mfs3_p0 \
    //host2/vol3/dat/mfs3_p1 \
    //host3/vol7/dat/mfs3_p2

$ m_mkfs my-mfs my_mfs_p0 my_mfs_p1 my_mfs_p2
```

M_MKDIR COMMAND

m_mkdir *url*

- Creates the named multidirectory. The url must refer to a pathname within an existing multifile system.

\$ m_mkdir mfile:my-mfs/subdir

\$ m_mkdir mfile://host2/tmp/temp-mfs/dir1

M_LS COMMAND

m_ls [*options...*] *url [url...]*

- Lists information on the file or directories specified by the urls. The information presented is controlled by the options, which follow the form of ls.

```
$ m_ls -Id mfile:my-mfs/subdir
$ m_ls mfile:/host2/tmp/temp-mfs
$ m_ls -I -partitions .
```

M_EXPAND

m_expand [*options...*] *path*

- Displays the locations of the data partitions of a multifile or multidirectory

```
$ m_expand mfile:mymfs
$ m_expand -native /path/to/the/mdir/bar
```

M_DUMP COMMAND

m_dump *metadata* [*path*] [*options ...*]

- Displays contents of files, multifiles, or selected records from files or multifiles, similar to View Data from GDE.

```
$ m_dump simple.dml simple.dat -start 10 -end 20
$ m_dump simple.dml -describe
$ m_dump simple.dml simple.dat -end 1 -print 'id*2'
$ m_dump help
$ m_dump -string 'string("\n")' bigex/acct.dat
```

M_CP COMMAND

m_cp *source dest*

m_cp *source [...] directory*

- Copies files or multifiles that have the same degree of parallelism. Behind the scenes, m_cp actually builds and runs a small graph, so it may copy from one machine to another where Ab Initio is installed.

\$ m_cp foo bar

**\$ m_cp mfile:foo **

mfile://OtherHost/path/to/the/mdir/bar

**\$ m_cp mfile:foo mfile:bar **

//OtherHost/path/to/the/mdir

M_MV COMMAND

m_mv *oldpath newpath*

- Moves a single file, multifile, directory, or multi-directory from one path to another path on the same host via renaming... does not actually move data.

\$ m_mv foo bar

\$ m_mv mfile:foo mfile:/path/to/the/mdir/bar

M_TOUCH COMMAND

m_touch *path*

- Creates an empty file or multifile in the specified location. If some or all of the data partitions already exist in the expected locations, they will not be destroyed.

\$ m_touch foo

\$ m_touch mfile:/path/to/the/mdir/bar

M_RM COMMAND

m_rm [*options*] *path [...]*

- Removes a file or multifile and all its associated data partitions.

```
$ m_rm foo
```

```
$ m_rm mfile:foo mfile:/path/to/the/mdir/bar
```

```
$ m_rm -f -r mfile:dir1
```

Partition & Departition components

Data can be partitioned using

- **Partition by Round-robin**
- **Partition by Key**
- **Partition by Expression**
- **Partition by Range**
- **Partition by Percentage**
- **Partition by Load Balance**

**Writes records to each partition evenly
Block-size records go into one partition
before moving on to the next.**

**Data may not be evenly distributed across
partitions**

**Does not guarantee even distribution across
partitions**

**Cascaded Filter by Expressions can be
avoided**

Increases data parallelism when connected single fan-out flow to out port

Percentage parameter set to “30 20”

Key range is passed to the partitioning component through its split port

Key values greater than 73 go to partition 2

Summary of Partitioning Methods

DEPARTIONING COMPONENTS

- **Gather**
- **Concatenate**
- **Merge**
- **Interleave**

- **Gather**
 - **Reads data records from the flows connected to the input port**
 - **Combines the records arbitrarily and writes to the output**
- **Concatenate**
 - **Concatenate appends multiple flow partitions of data records one after another**
- **Merge**
 - **Combines data records from multiple flow partitions that have been sorted on a key**
 - **Maintains the sort order**

Summary of Departmenting Methods

LOOKUP

- **Serial or Multifiles**
- **Held in main memory**
- **Searching and Retrieval is key-based and faster as compared to files stored on disks**

- associates key values with corresponding data values to index records and retrieve them
- **Lookup parameters**
 - **Key**
 - **Record Format**

NOTE: Data needs to be partitioned on same key before using lookup loc

DB COMPONENTS

- **db_config_utility** : Generate interface file to the database
- **Input Table**
 - unloads data records from a database into an Ab Initio graph
 - **Source** : DB table or SQL statement to SELECT from table
- **Output Table**
 - loads data records into a database
 - **Destination** : DB table or SQL statement to INSERT into table
- **Update Table**
 - executes UPDATE or INSERT statements in embedded SQL format to modify a DB table

RUN PROGRAM COMPONENT

Use Run Program for any program that works with standard input/output, such as many shell filters (for example, grep)

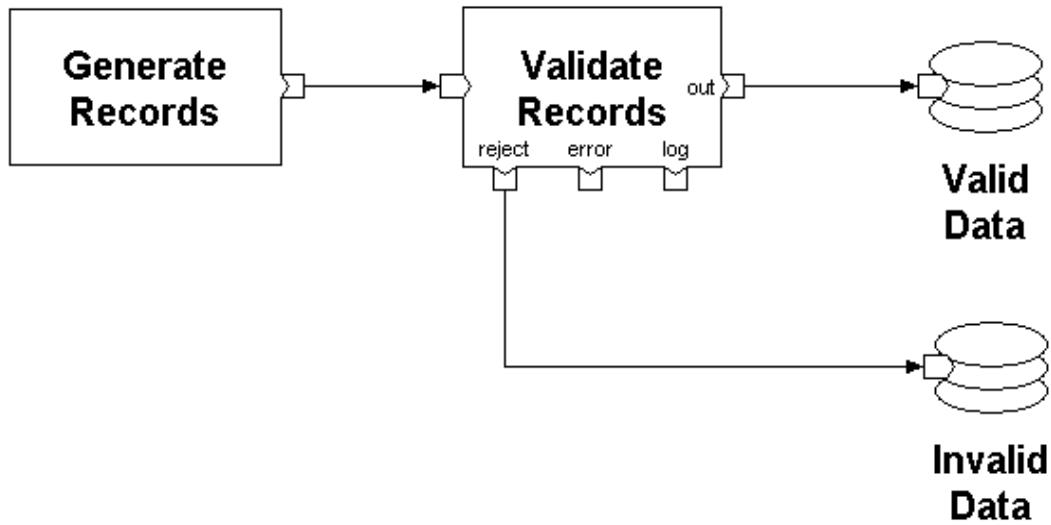
- **Use the commandline parameter to specify the program name and arguments**
- **Standard input (stdin) is tied to input port**
- **Standard output (stdout) is tied to output port**
- **Standard error (stderr) comes out of graph's stderr only if program fails**

VALIDATE COMPONENTS

The Validate Records component uses the `is_valid()` function to check each field of the input data records to determine if the value in the field is:

- Consistent with the data type specified for the field in the input record format
- Meaningful in the context of the kind of information it represents

EXAMPLE



A Deadlock happens when

**when the graph stops progressing because of mutual dependency
of data among components**

**Identified when record count in does NOT change over a period of
time**

Deadlock can be avoided using

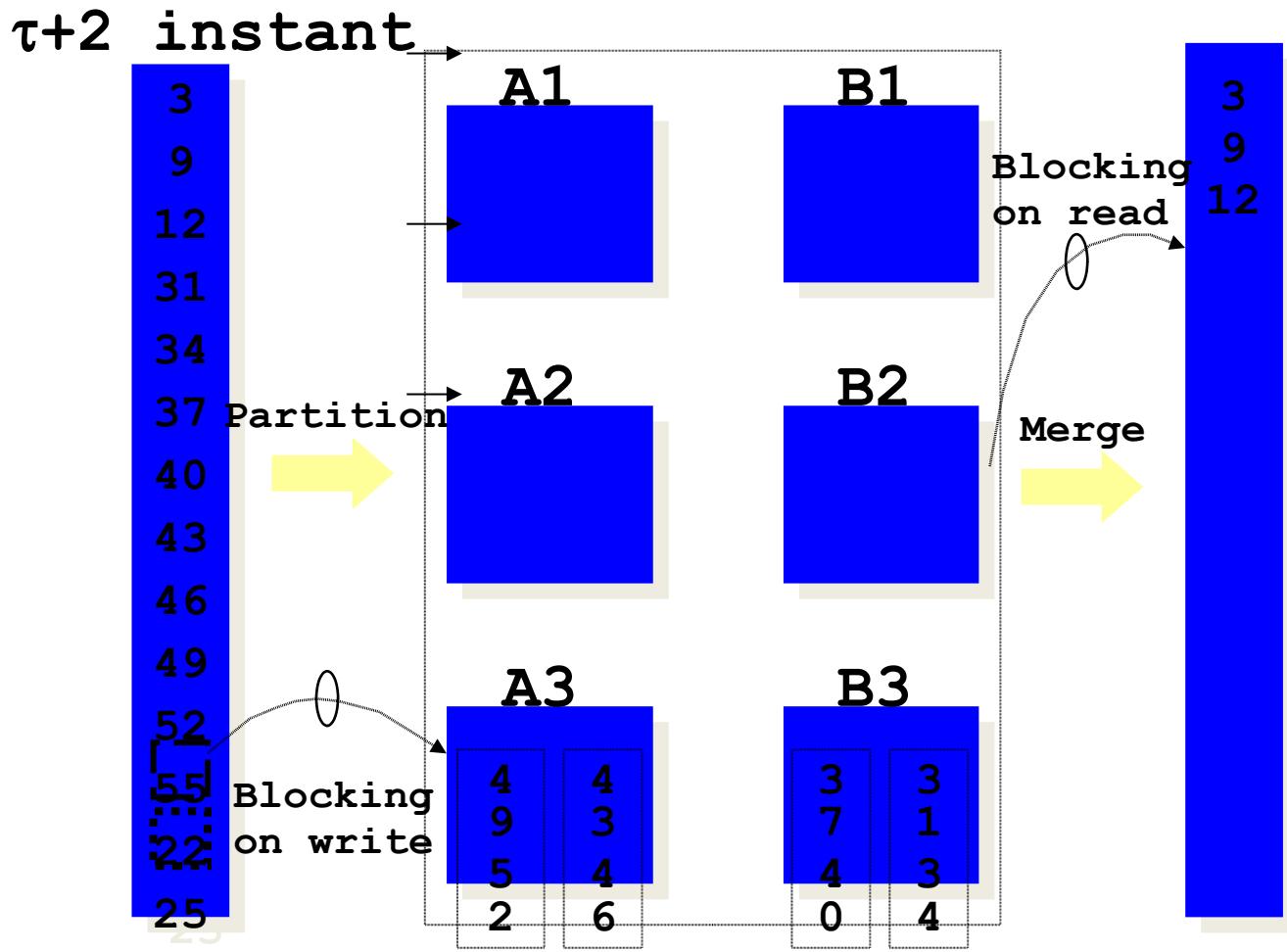
Phasing

Checkpointing

Components likely to cause deadlock

Input/output buffer size 2

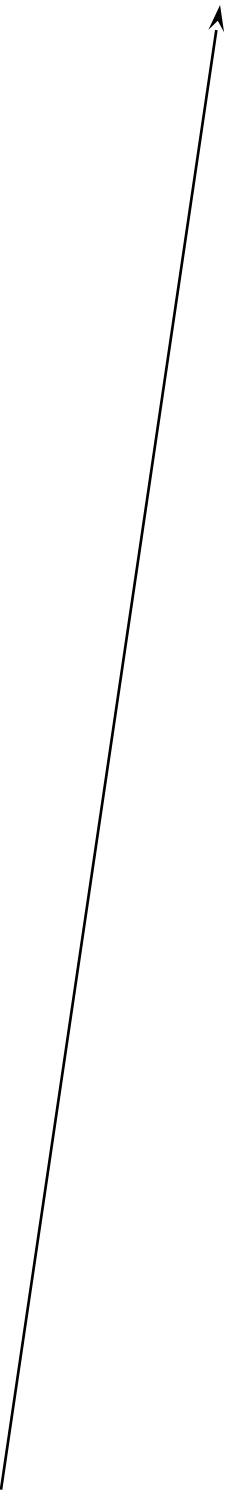
Input/output buffer size 2

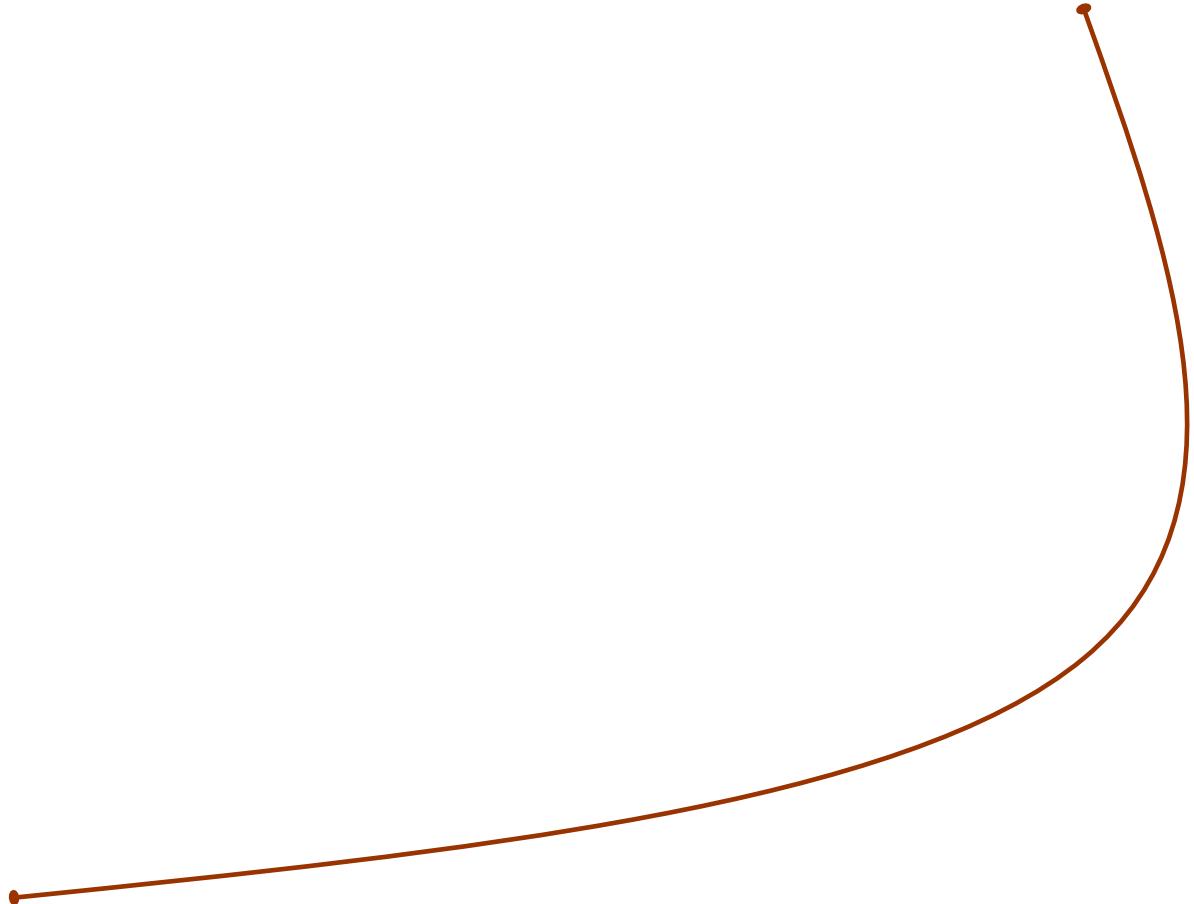


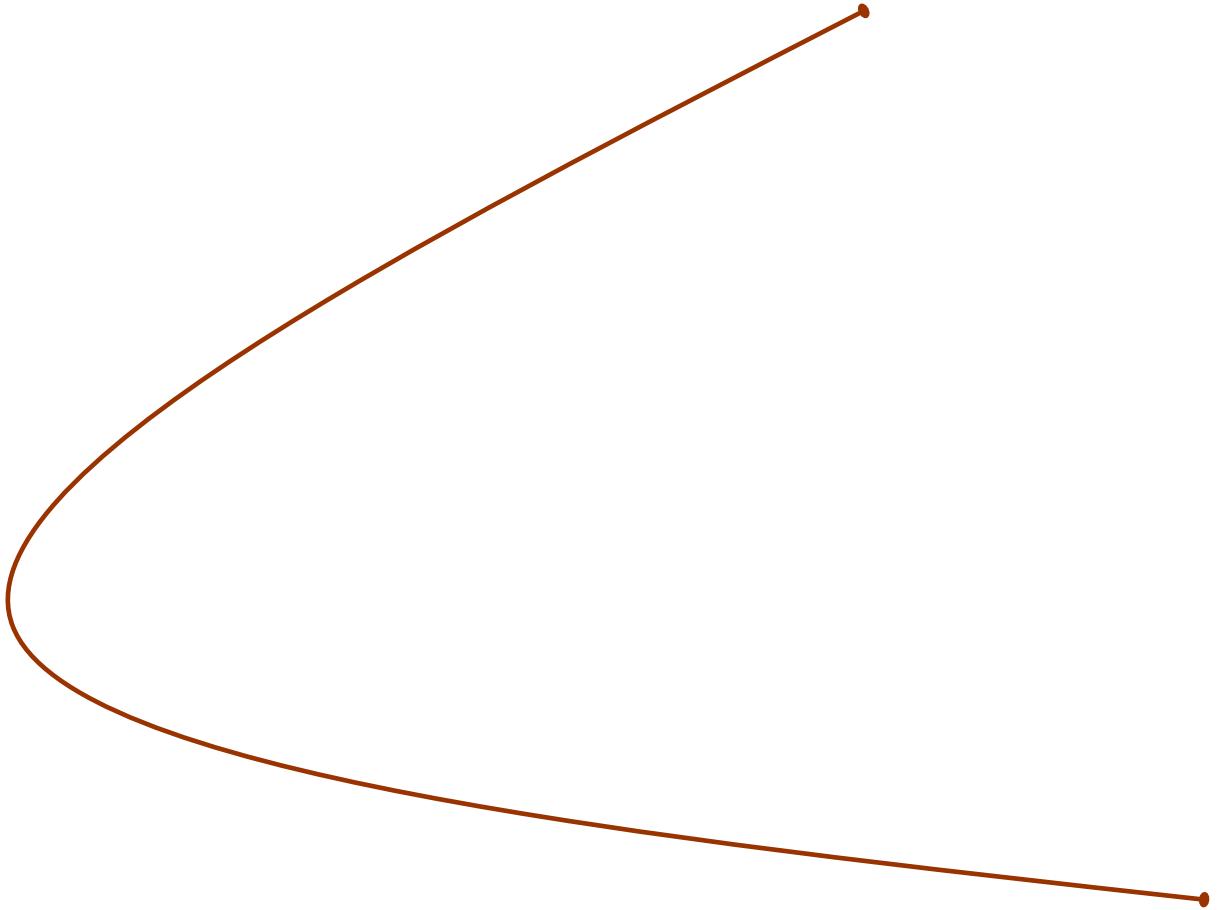
AVOID DEADLOCK

- Use Concatenate and Merge carefully
- Use flow buffering (the GDE Default).
- Insert a phase break before the departitioner.
- Don't serialize data unnecessarily
- Repartition instead of department.

LAYOUT







file on Host W

Database components
can use the same layout
as a database table

IMPROVE PERFORMANCE OF GRAPH

1. Use limited number of components
2. Minimize number of SORT component.

- 3. Minimize sorted join component, if possible replace them by in-memory join.**
- 4. Use multifile.**
- 5. Minimize regular expression functions (like re_index) in the transfer functions**

END OF SESSION 5.

START WITH SESSION 6 [FROM AB-INITIO MATERIAL]

Overview: What is the Enterprise Meta>Environment™? [EME]

EME – VERSION CONTROL TOOL.

Using the EME...

...from the GDE

...from the shell

...from the web and GDE

...from the shell

meta>environment: an environment for metadata

Data about data.” Record formats, database ddl, field/column definitions, business rules, business processes, job statistics, version history, etc.

Everything but the data itself.

meta>environment:

an environment for metadata

- ***organization***
- ***management***
- ***use***

*storage
manipulation
analysis*

TWO CLASSES OF METADATA

Ab Initio-related

- graphs, dml, xfr, db, etc.

= “Technical metadata”

Everything else

= “Business metadata”

CLASSIFICATION OF PROJECT

Separate technical metadata
by type

db - database-related

dml - record formats

mp - graphs

run - deployed scripts

xfr - transforms

ORGANIZE BY SUBDIVISION

Divide technical metadata
into “projects”
conceptually
for logical coherence
organizationally

Technical metadata EME uses
for manageability

- Central Metadata Storage
- Versioning
- Source code control
- Analysis
- Job status

EME ...from the developer's perspective

METADATA FOR PROJECTS

project metadata:
parameters

name-value pair

COMMON PROJECTS LOCATION
logical-physical mapping

Public and private project

Any project can be included-

but not all should be!

- public projects
 - intended to be included
- private projects
 - not to be included

core concepts

A “sandbox” is a user’s personal workspace:

Graphs and metadata are “checked out” from the EME into a sandbox.

New and modified graphs and metadata are then “checked in” to the EME.

SANDBOX STRUCTURE

Referring Parameters

\$AI_DB

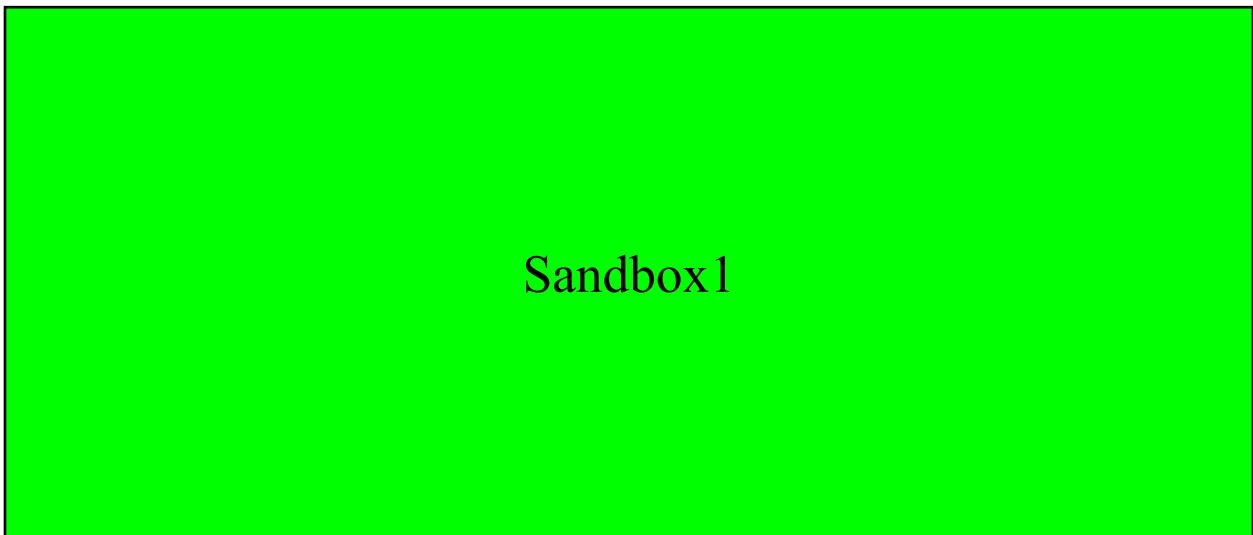
\$AI_DML

\$AI_MP

\$AI_RUN

\$AI_XFR

SANDBOX – PROJECT CORRESPONDANCE



USERS

Each user

- has their own sandbox(es)
- one to a project

Project may have multiple copies in file system.

SANDBOX PROJECT CORRESPONDANCE

Sandboxes are distinguished from projects.

They may contain objects that are either

- not checked in to source control

- currently being modified
- the same as in project

EME FILE SYSTEM CORRESPONDACE

File system

FROM EME TO GDE

GDE USES OF EME

Developers - Source Code Control:

- Check Out
- Check In
- Analyze Dependencies

Administration

- Create Projects
- Edit Projects

Configuring the gde

PROJECT → SETTINGS

Click the **Connect** button.
Save your settings as
Normal.aip.

Set your **Run>Settings** to “Normal”

Fill in host, login, and password

Leave Host directory blank

Checking Out

PURPOSE OF CHECKOUT

Check Out

- to look at (latest) version
- before modifying

Best Practice: Check Out

- project at a time

FILE SYSTEM – EME RELATIONSHIP

The EME stores objects under source control.

Your work area is called a “sandbox.”

Check Out objects from the EME to your sandbox.

CREATE A NEW SANDBOX

Browse for a sandbox directory **Next>. Check Details. Do Checkout.**

CHECKOUT INDIVIDUAL COMPONENTS

Check Out applies to graphs and other objects as well.

During Check Out, simply browse to the object you want to checkout.

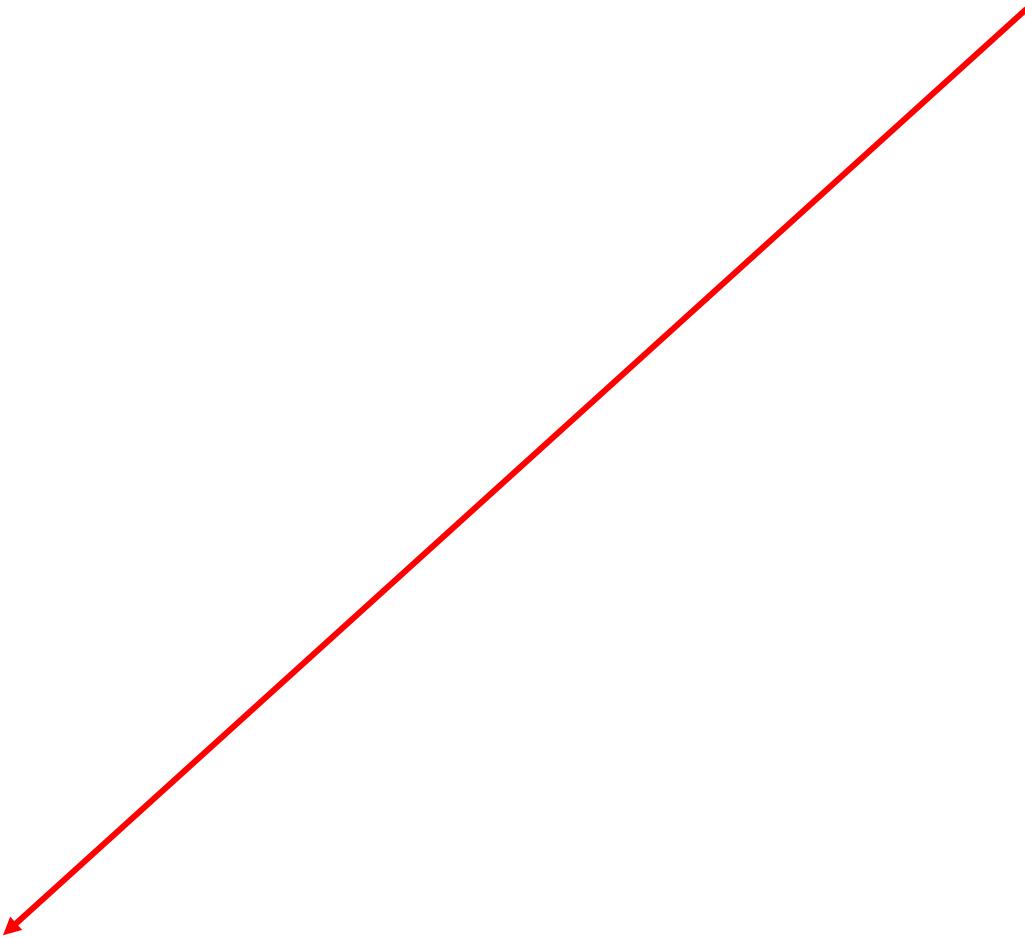
Modifying a graph

DEMONSTRATION / MODIFICATION

Modify the graph

EDW_populate_from_catalog_cust_info.mp

Check In your changes.



LOCKING

Lock objects before modifying them.

You can't lock if someone else already has.

Locking controls access.

CONFLICT MANAGEMENT

No branching or merging.

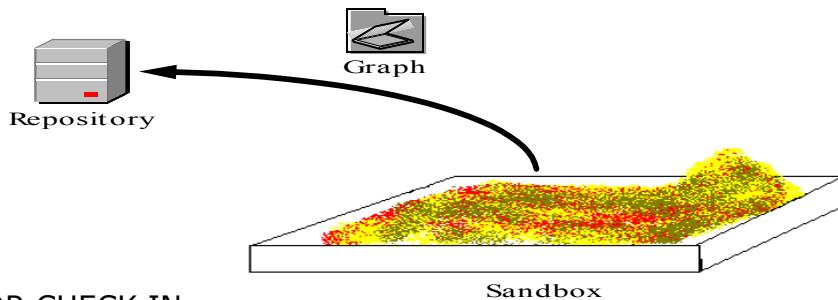
Simultaneous editing blocked.

Lock breakable by administrator.

Modify a graph you have opened, e.g.

EDW_populate_from_catalog_cust_info.mp

CHECK IN



NEED FOR CHECK IN

Check In

- for source code control
- to share code

Best Practice: Check In

- one graph at a time

Project

Modify a graph, e.g.

EDW_populate_from_catalog_cust_info.mp

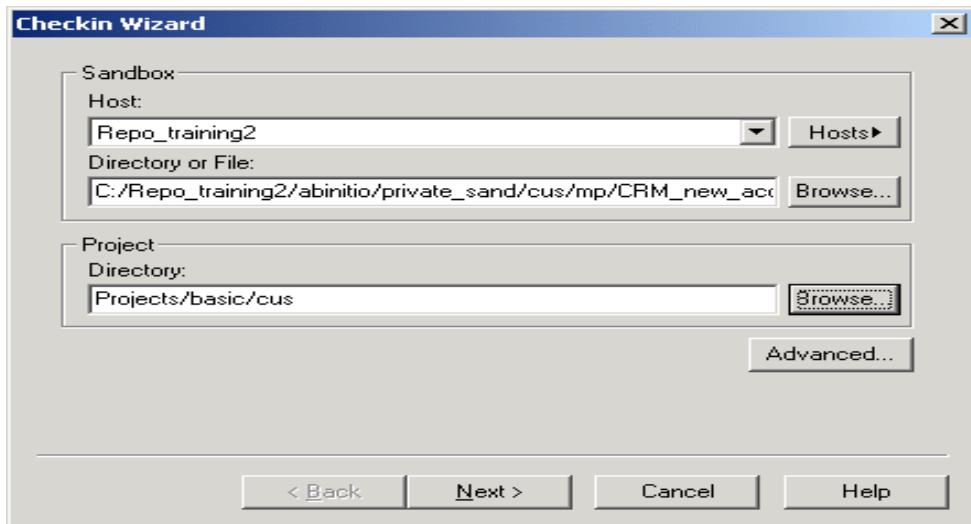
and Check In.

Assign tags to label version of an object.

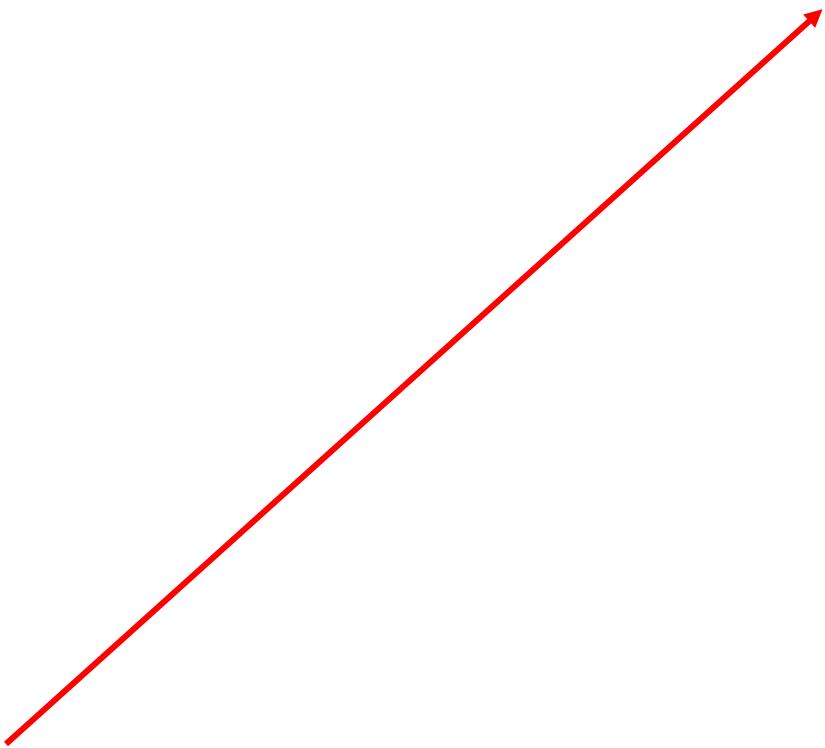
Enables

- differencing
- reverting
- promoting

CHECKIN WITH TAGS



TAGGING



TAGS APPLY TO RELATED OBJECTS

When you tag a graph, all its related dml and xfr files are also tagged.

Apply the same tag to more than one graph to identify a set.

An object can have more than one tag

You can apply a tag to a set of tags to create a configuration.

Configurations identify objects as part of a release for promotion.

CHECKIN PROJECT AND GRAPH

Check In applies to individual files, as well as whole projects.

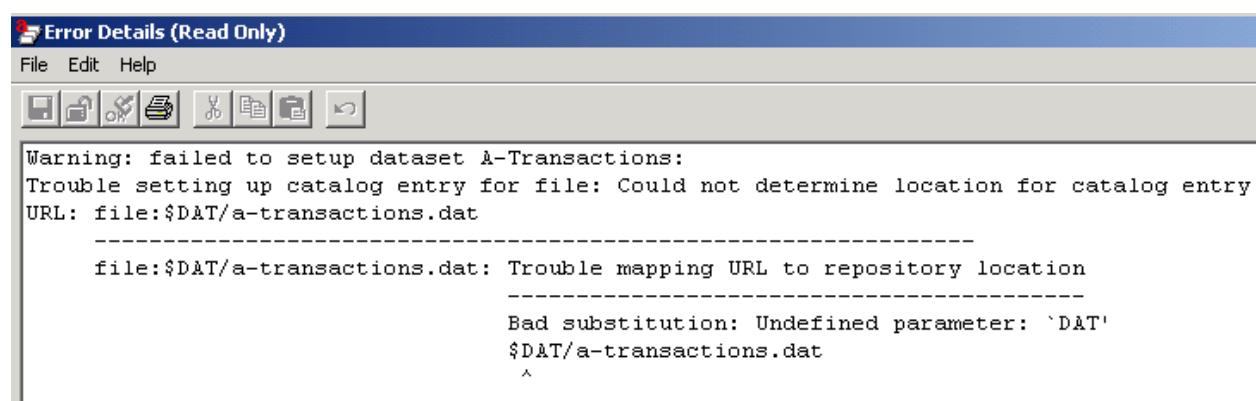
Simply browse to the appropriate object during Check In.

CHECKIN IS SELECTIVE

Check In only applies to objects that have been modified.

Its okay to Check In a graph even if only a dml file has changed

DEPENDENCY AND ANALYSIS



The screenshot shows a software interface titled "Error Details (Read Only)". The menu bar includes "File", "Edit", and "Help". Below the menu is a toolbar with several icons. The main area displays error messages related to dataset setup and URL mapping:

```
Warning: failed to setup dataset A-Transactions:  
Trouble setting up catalog entry for file: Could not determine location for catalog entry  
URL: file:$DAT/a-transactions.dat  
-----  
file:$DAT/a-transactions.dat: Trouble mapping URL to repository location  
-----  
Bad substitution: Undefined parameter: 'DAT'  
$DAT/a-transactions.dat  
^
```

DEPENDENCY ANALYSIS

The EME understands graphs.

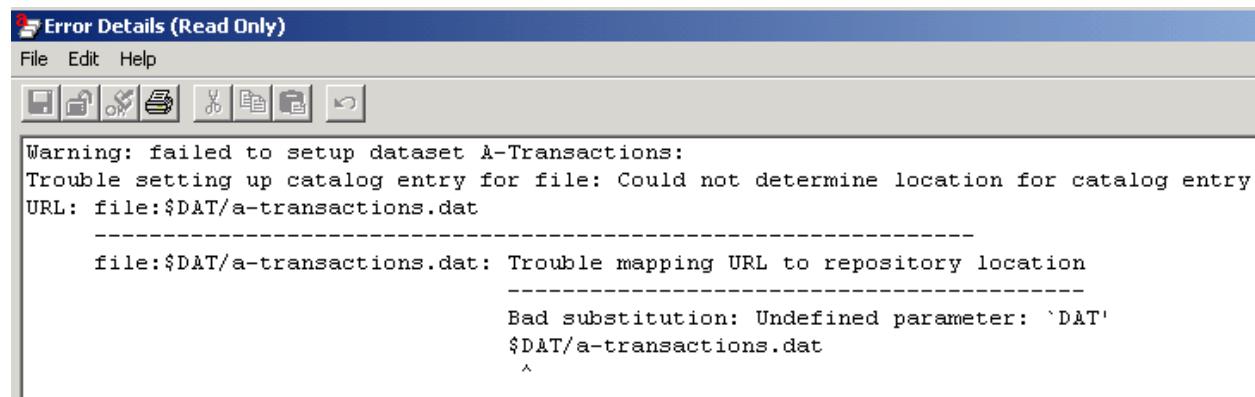
Dependency analysis builds connections between objects in the EME.

This enables the EME to deduce the flow of data between graphs.

Parameters for directories are used to construct a mapping between metadata locations in a sandbox and locations in the EME.

This is the key to dependency analysis.

Warnings result when the GDE cannot identify the mapping because a parameter is not defined.



The screenshot shows a software interface titled "Error Details (Read Only)". The menu bar includes "File", "Edit", and "Help". Below the menu is a toolbar with various icons. The main area displays a warning message:

```
Warning: failed to setup dataset A-Transactions:  
Trouble setting up catalog entry for file: Could not determine location for catalog entry  
URL: file:$DAT/a-transactions.dat  
-----  
file:$DAT/a-transactions.dat: Trouble mapping URL to repository location  
-----  
Bad substitution: Undefined parameter: `DAT'  
$DAT/a-transactions.dat  
^
```

WARNING AND RESOLUTION

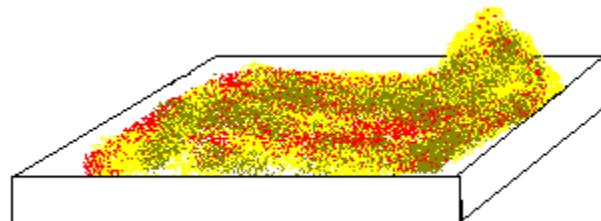
At Check In, warnings issued

- for undefined parameters
- for invalid dml or xfrs

Resolve by

- defining parameters
- fixing errors

SANDBOX



A “sandbox” is a user’s personal workspace:

Graphs and metadata are “checked out” from the EME into a sandbox.

New and modified graphs and metadata are then “checked in” to the EME.

SANDBOX STRUCTURE

Referring Parameters

\$AI_DB

\$AI_DML

\$AI_MP

\$AI_RUN

\$AI_XFR

SANDBOX PROJECT CORRESPONDANCE

Sandbox1

USERS

Each user

- has own sandbox(es)
- one to a project

Project may have multiple copies in file system.

ACCESS CONTROL

To view graphs in GDE,

you must Check Out to a sandbox.

Permissions

- allow read-only access
- block access

CREATE A NEW GRAPH

Create a new graph with File>New.

The Graph is not locked but is editable since it is not known to the EME.

The GDE does not yet associate your graph with a sandbox.

Save it to a sandbox to allow the parameters to be used.

USING PARAMETERS

DEMONSTRATION AND PARTICIPATION

Look at the sandbox parameters of

/Projects/basic/cus.

Add a parameter REPORT with value \$AI_MFS/report.dat.

PROJECT PARAMETERS

Every Ab Initio project uses logical names for physical values that may change-

- between runs
- between project lifecycle stages
- between machines.

Use parameters

- to enhance portability
- to promote flexibility
- to simplify communication
- Examples:
 - Metadata directories (dml, xfr)
 - Data directory names (serial or MFS)
 - Database Names and install locations

Referring Parameters

\$AI_DB

\$AI_DML

\$AI_MP

\$AI_RUN

\$AI_XFR

EDITING PARAMETERS

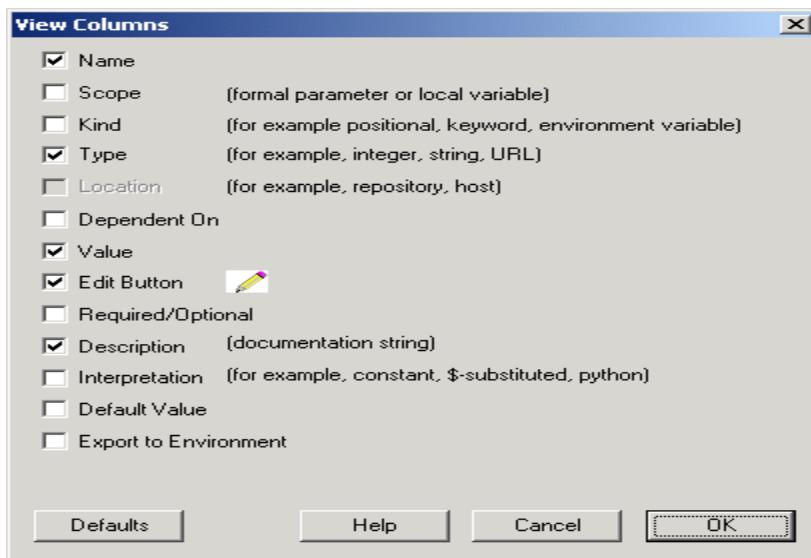
Use Project>Edit Sandbox>Parameters to access parameters

Most parameters have

scope = local

type = string

VIEW COLUMNS



EME VS SANDBOX PARAMETERS VALUES

Parameters values may differ between the EME and sandbox.

Example: PROJECT_DIR is root of project

Sandbox: PROJECT_DIR /users/jsmith/lesson

EME: PROJECT_DIR /Projects/lesson

Example: Sandbox value

- Directory path,
e.g. AI_DML for \$PROJECT_DIR/dml

may resolve to

/users/jsmith/lesson/dml

Example: EME value

- Directory path,
e.g. DML for \$PROJECT_DIR/dml

may resolve to

/Projects/lesson/dml

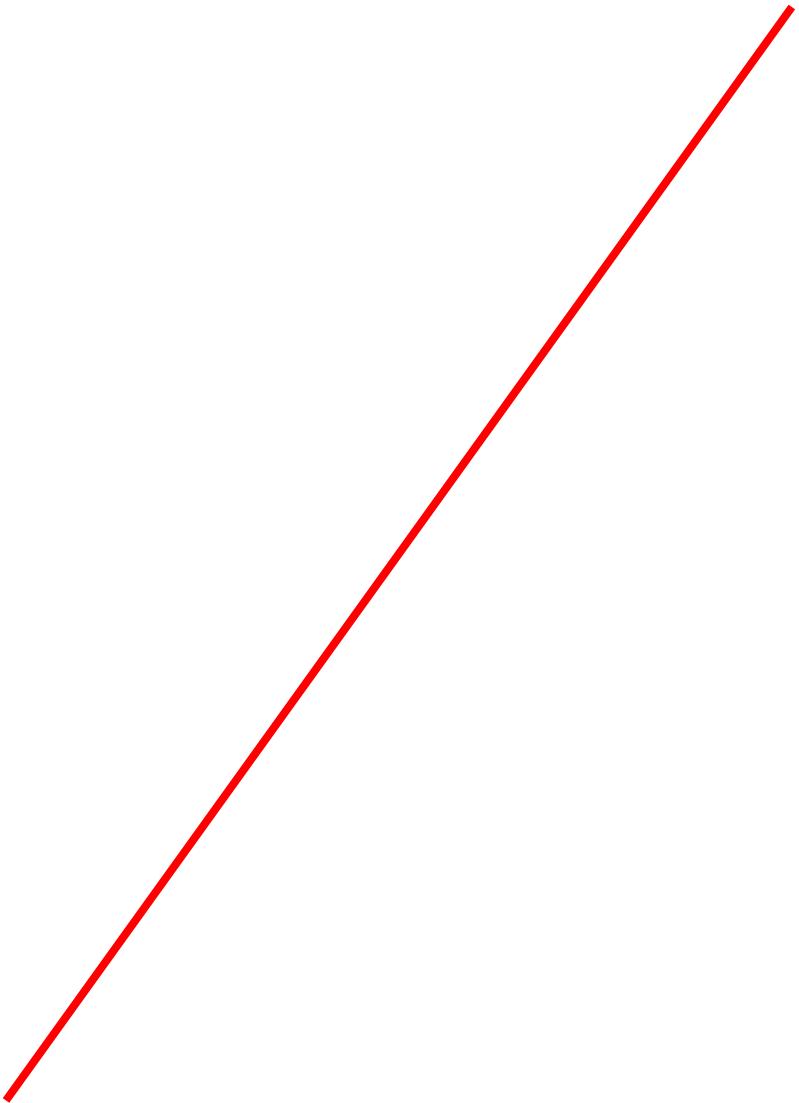
COMMON PROJECTS

Include another project to use its parameters.

The included project is called a “common” project.

Any project can be “common.”

NO CIRCULAR REFERENCES



Put shared metadata
in a project that
other projects include.

Datasets that are shared
must be defined in
included projects.

USE EME FROM SHELL

INVOKE THE ENVIRONMENT

Two ways: In a sandbox

. ab_setup_project.ksh \$PWD

or:

. <path to stdenv sandbox>/project-start.ksh

set_environment \$PWD

AIR COMMANDS

Interact with the EME from the shell using air commands

Classes of commands

- repository
- project
- sandbox

- object

AB_AIR_ROOT

Specify the default EME with the environment variable

AB_AIR_ROOT

For example,

```
export AB_AIR_ROOT=$PWD/repo_training
```

EME listing:

```
air ls -l <rpath>
```

Information about a command

```
air help <command>
```

For example,

```
air help ls
```

air repository commands

Create a new EME

```
air repository create <name>
```

Remove an EME

```
air repository destroy
```

air repository backup allows any backup command to be run on a locked EME

air repository create-image saves an entire EME in a portable text format

air repository load-from-image creates a EME from a saved image file

air project, sandbox and object commands

INFO DISPLAY

Project information

air project show <project-name>

Parameter evaluation

air project parameter <proj> -eval <parm>

AIR PROJECT SHOW

Specify where non-project-related items are stored.

Check In

air project import <proj> -basedir <root>

Check Out

air project export <proj> -basedir <root>

Many other options.

ANALYSIS MAINTAINANCE COMMANDS

air project show-queue shows contents of queues

air project dequeue removes graphs from the dependency analysis or translation queues

air project enqueue adds graphs to the dependency analysis or translation queues

SANDBOX MAINTAINANCE COMMANDS

air sandbox create creates a new sandbox

air sandbox parameter edits and views sandbox parameters

air sandbox lock locks a file in a sandbox

air sandbox revert unlocks a file in a sandbox, and restores it to the latest checked in version

air sandbox detach uncouples a sandbox from an EME project

PROMOTION

Save to a portable format

air save <file> <rpath> <...>

Load from a portable format

air load <file>

GENERAL OBJECT COMMANDS

- air mvmove an object
- air cp copy an object
- air ln create a symbolic link
- air rm remove an object

Tagging

A "tag" is a label which marks a set of specific versions of repository objects.

PROMOTION

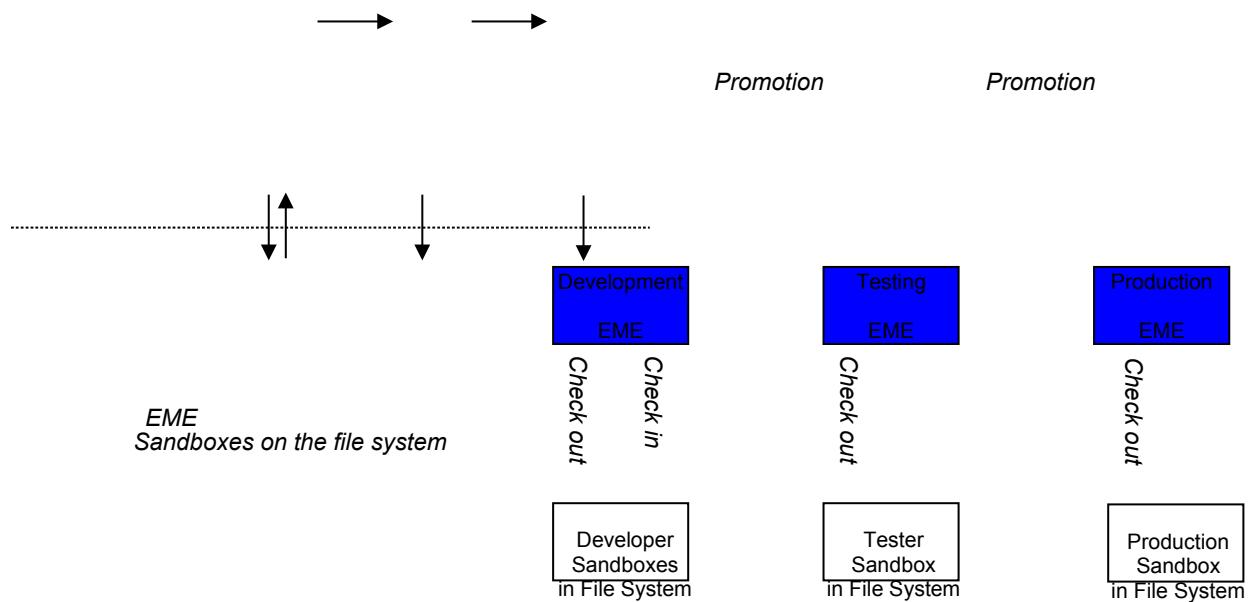
Promotion refers to the movement of graphs and associated code from one environment to another.

The EME automates much of promotion and mitigates many of the risks.

PRIMARY OBJECTS

Primary objects (usually graphs) and associated secondary objects (DML, XFR, shell scripts, etc) move together.

Ensures that errors are not introduced into applications as a result of promotion



DEVELOPER VIEW OF PROMOTION

Developers

- checkin and checkout code
- run Dependency Analysis to prepare for promotion

- Dependency Analysis enables the EME to understand the required objects for a given graph.

DEVELOPER VIEW OF PROMOTION

Association of related objects is usually automatic through dependency analysis.

Can specify manually using:

air requires <rpath> [...]

e.g. AB_INCLUDE_FILES

TAGS AND CONFIGURATIONS

Tag an object and its required files using

air tag create <tag> <object>

If you tag a project, all objects in the project are tagged.

air tag ls [-p | -e] <tag>

lists the objects designated by the tag.

-p lists the “primary” objects,

-e lists the required files as well.

air tag import-configuration <config file> <EME path>

imports a configuration file into the EME

air tag tag-configuration <tag> <config file>

creates a tag based on the configuration file

PROMOTION Two approaches:

- Check Out tagged objects to staging sandbox, detach sandbox (if needed), Check In to new EME
- air save/air load commands for creating portable formats

AIR SAVE COMMANDS

The air save command exports selected objects and relationships from the EME into a flat file.

- Dependency analysis information is saved for graphs
- Related objects and relationships optionally can be saved
- Project settings can optionally be saved

AIR LOAD

The air load command imports data saved to a flat file with air save.

Usage:

```
air load <filename>
```

VALIDATION PROMOTION

Validate promoted graphs by running dependency analysis after loading into the destination EME

Other graphs can be validated as well to be sure that they haven't been broken by the new changes introduced

AIR PROJECT ANALYSE DEPENDENCIES

runs dependency analysis on the server without need for the GDE.

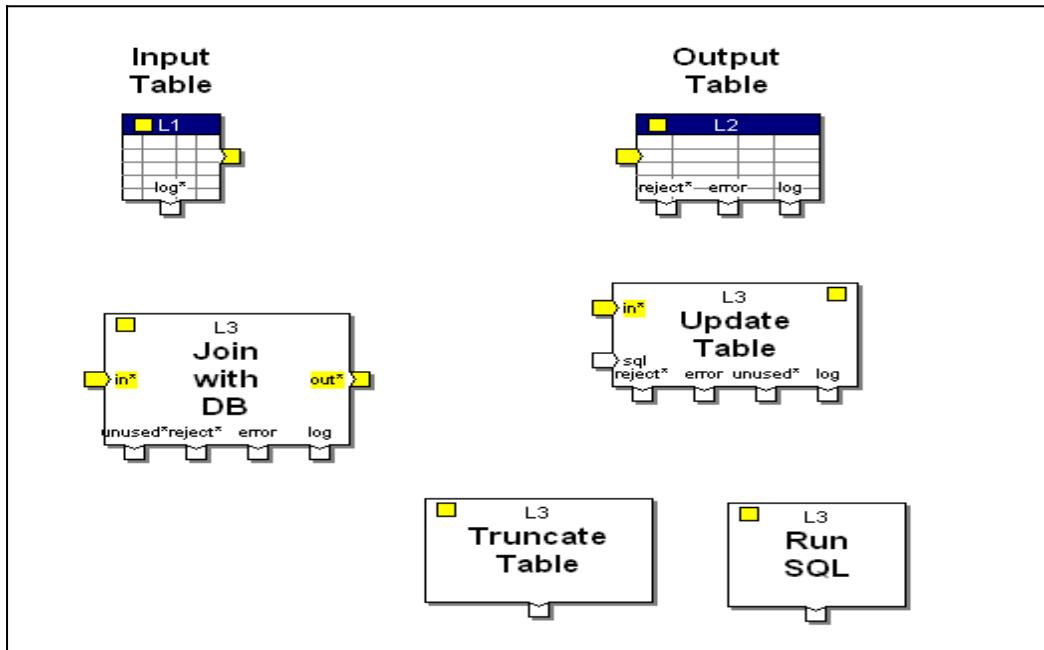
<END OF SESSION 6>

Session 7

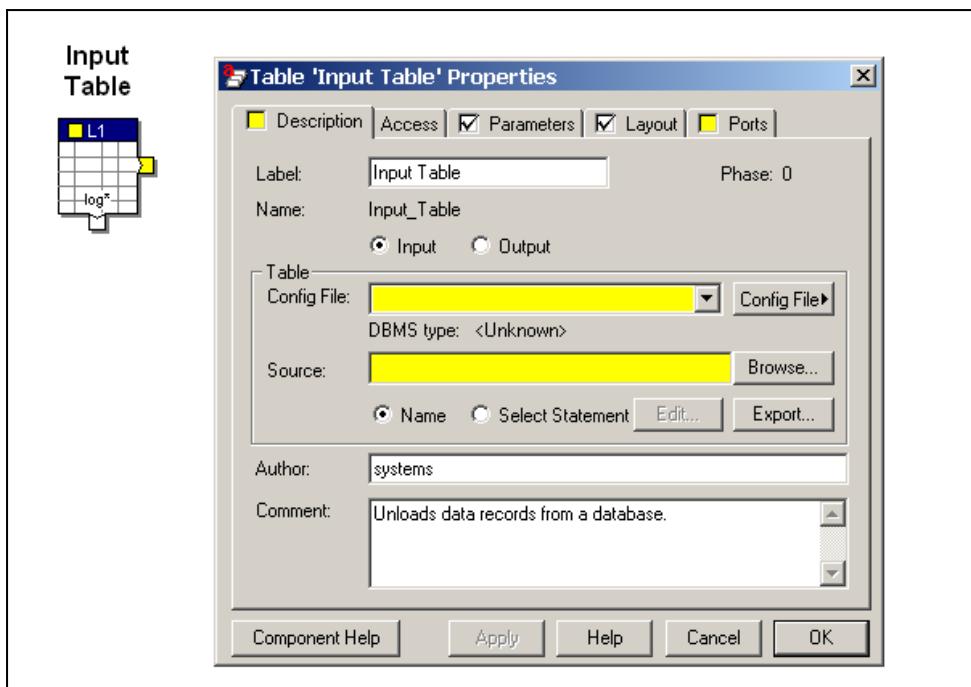
DB COMPONENTS

- **db_config_utility : Generate interface file to the database**
- **Input Table**
 - **unloads data records from a database into an Ab Initio graph**
 - **Source : DB table or SQL statement to SELECT from table**
- **Output Table**
 - **loads data records into a database**

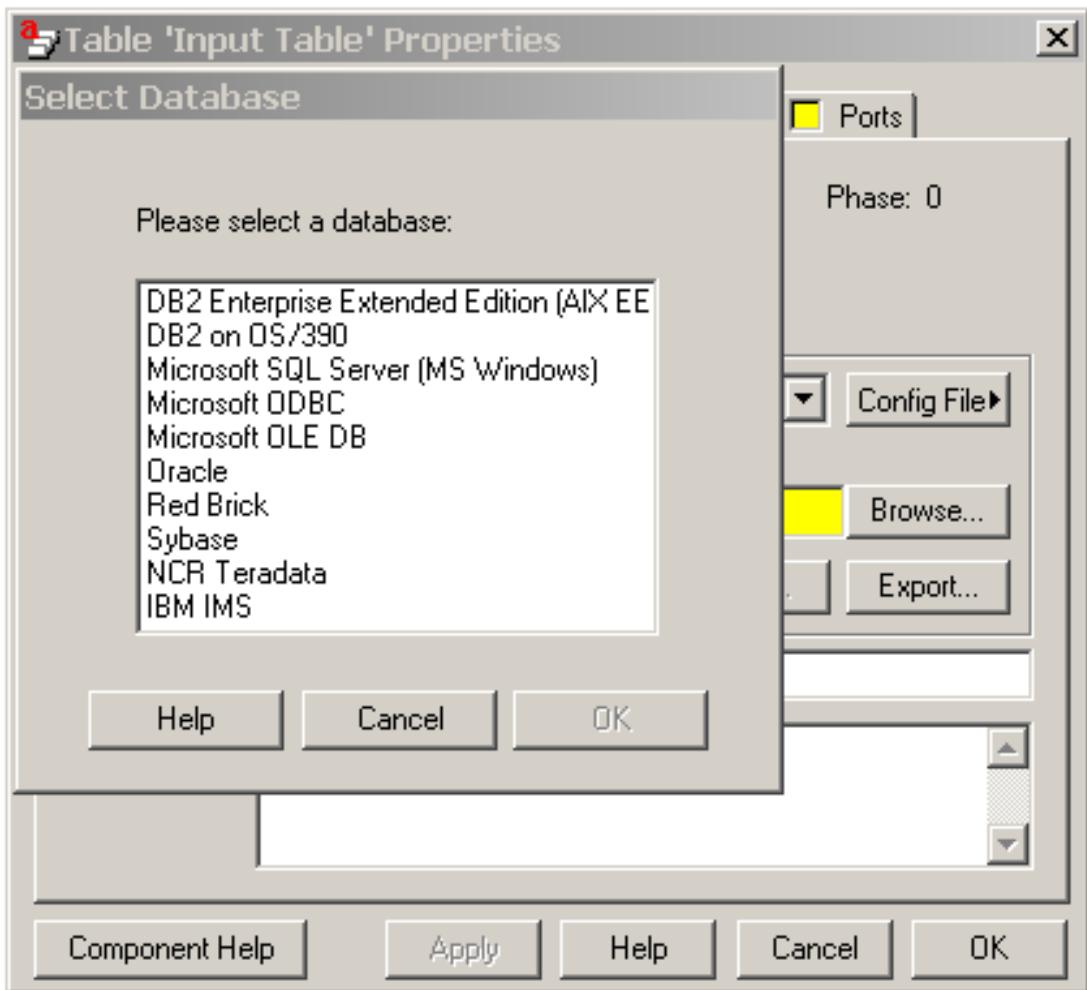
- **Destination : DB table or SQL statement to INSERT into table**
- **Update Table**
 - **executes UPDATE or INSERT statements in embedded SQL format to modify a DB table**
- **Truncate Table**
 - **deletes all the rows in a specified DB table**
- **Run SQL**
 - **executes SQL statements in a DB**



DB COMP REQUIRE A DATABASE CONFIG FILE [DBC]



SELECT CONFIG FILE → NEW TO CREATE A NEW CONFIG FILE



ABINITO DEFINED / GENERATED TEMPLATE FOR THE SAME

Edit Database Configuration - Microsoft SQL Server (MS Windows) *

File Edit Help

Microsoft SQL Server Database Configuration Information

You need to edit this file once, after you create

it. This file contains the information that tells

the Ab Initio (tm) IDB database components

how to connect to your Microsoft SQL Server database.

##

This file must always be named with the extension .dbc and

should be saved in the host directory where the graph

runs or in the %AB_HOME%\config\db directory.

##

Each line of this file must have the following form:

<tag>: <value>

##

Each tag must be at the start of a line and be

immediately followed by a colon (:). All white space

characters, such as blanks and tabs, are ignored.

##

This file also contains information about assigning

and editing the values of tags.

##

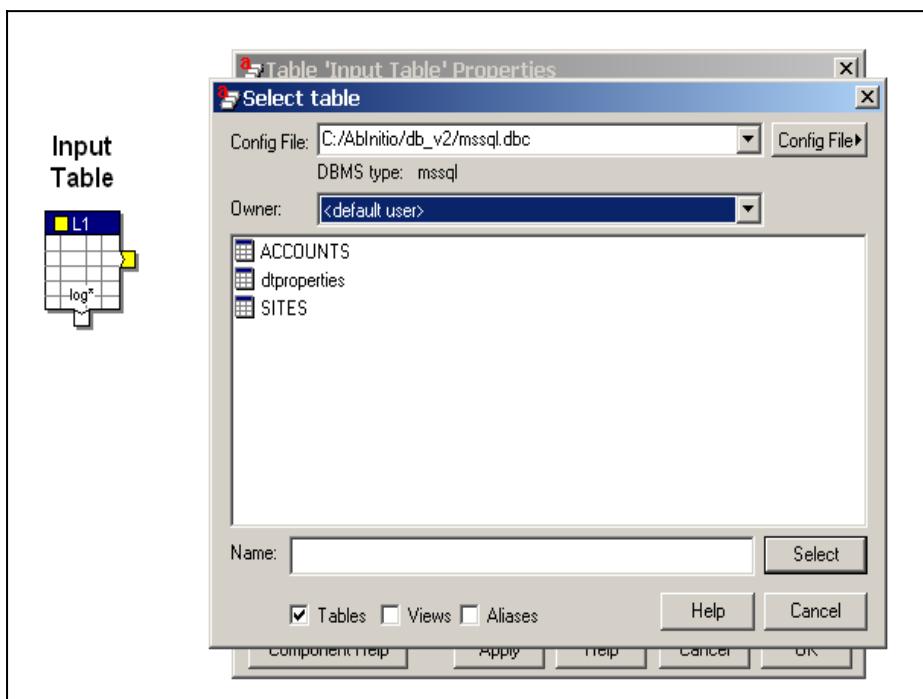
Pound signs at the beginning of a line denote

comments. The IDB database components ignore these

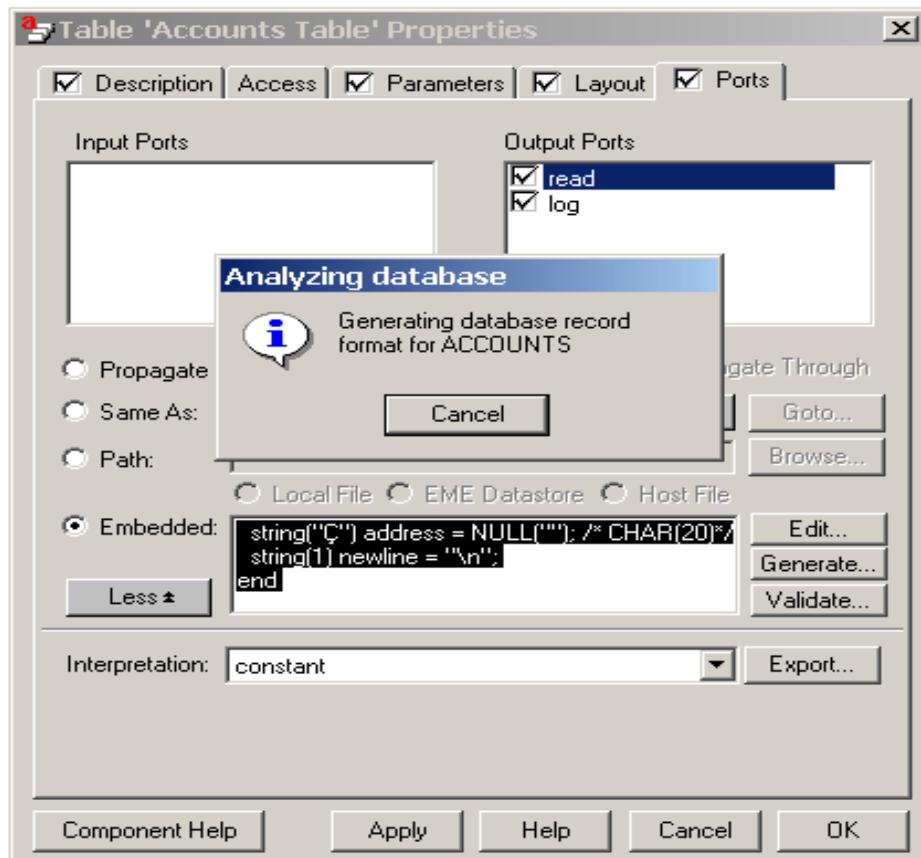
lines.

#####

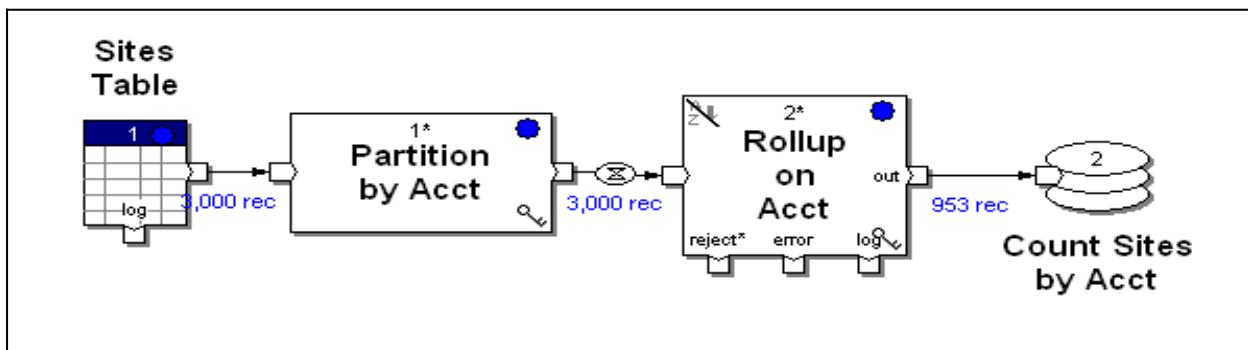
Once the dbc file is established one may browse the db instance



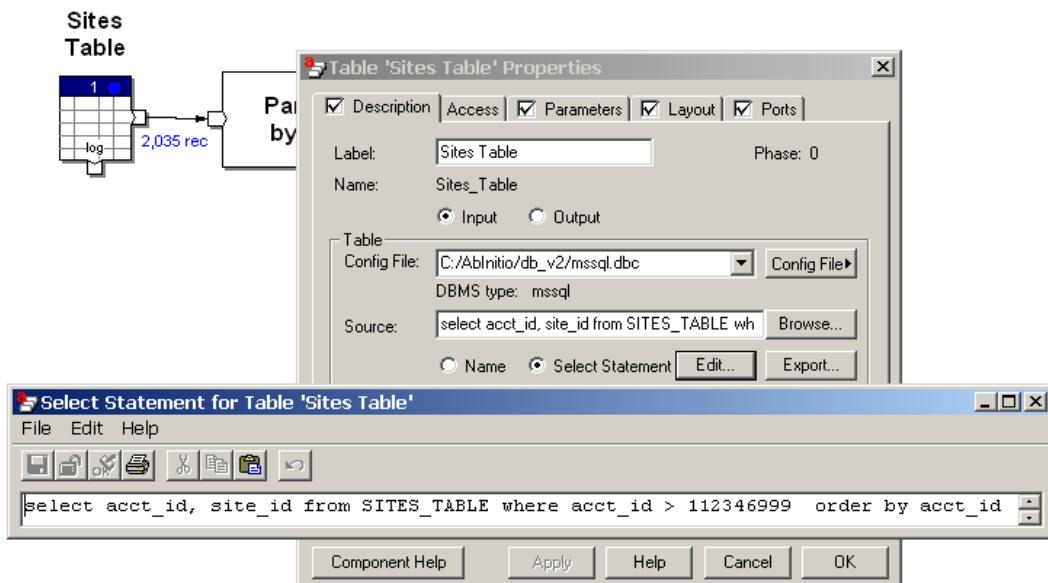
Clicking OK or Generate DML on Ports Tab will generate dml



Once connected, use as any data component



Can specify a select statement rather than a table



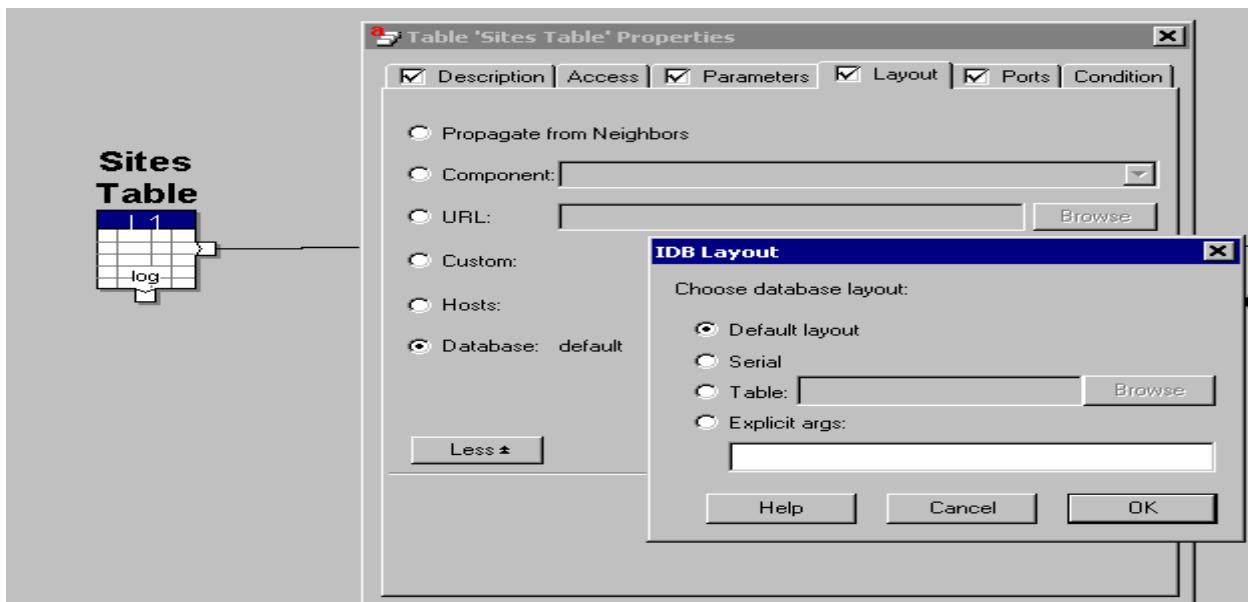
The select statement will determine what dml is generated

The screenshot shows a "Text Editor - Record Format for port 'read' of Table 'Sites Table'" window. The window has a toolbar with icons for file operations like Open, Save, Print, etc. The main area displays the generated DML code:

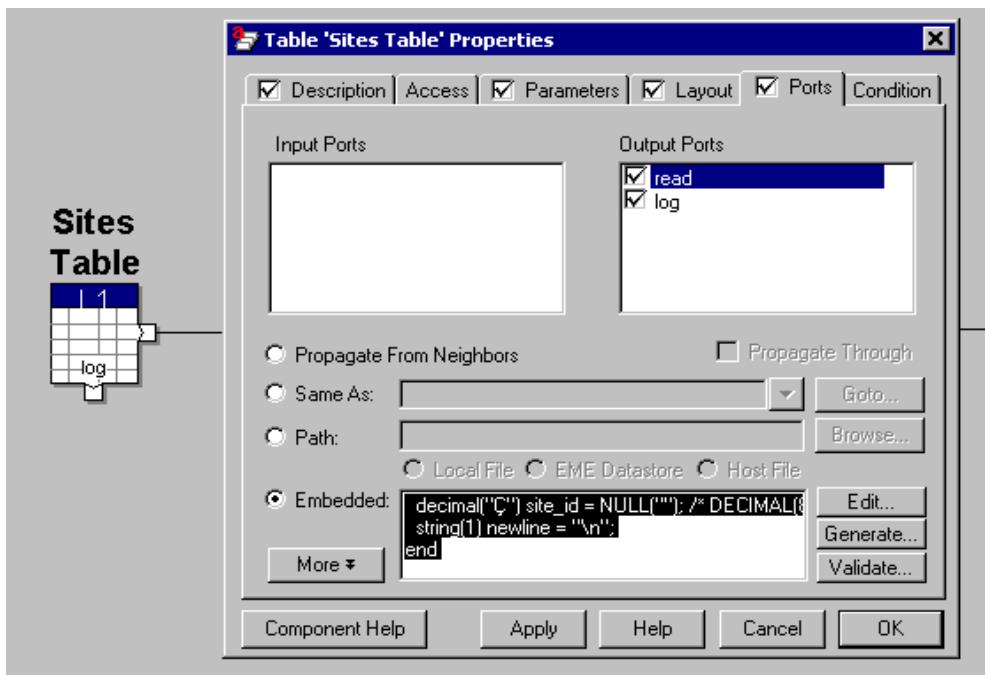
```
/* DML Generated for SQL: select acct_id, site_id from SITES_TABLE order by acct_id
 * On: Fri Nov 22 14:25:13 2002

*/
record
    decimal("Q") acct_id = NULL(""); /* DECIMAL(9) */
    decimal("Q") site_id = NULL(""); /* DECIMAL(8) */
    string(1) newline = "\n";
end
```

PROPERTIES Layout



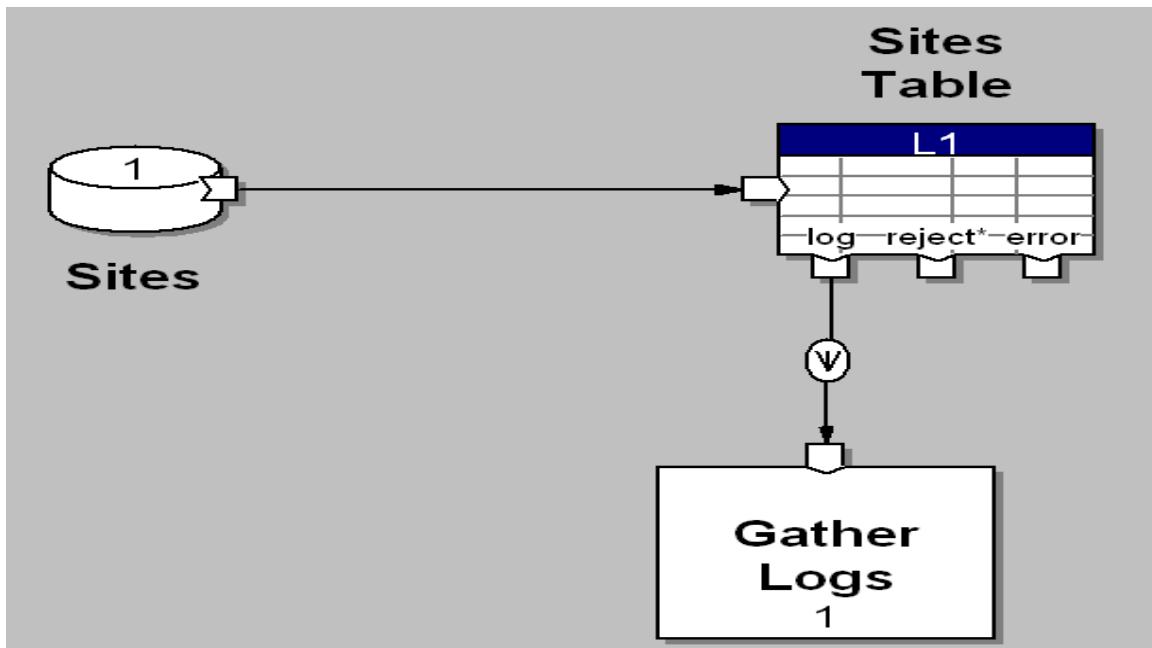
Properties port



Properties log port

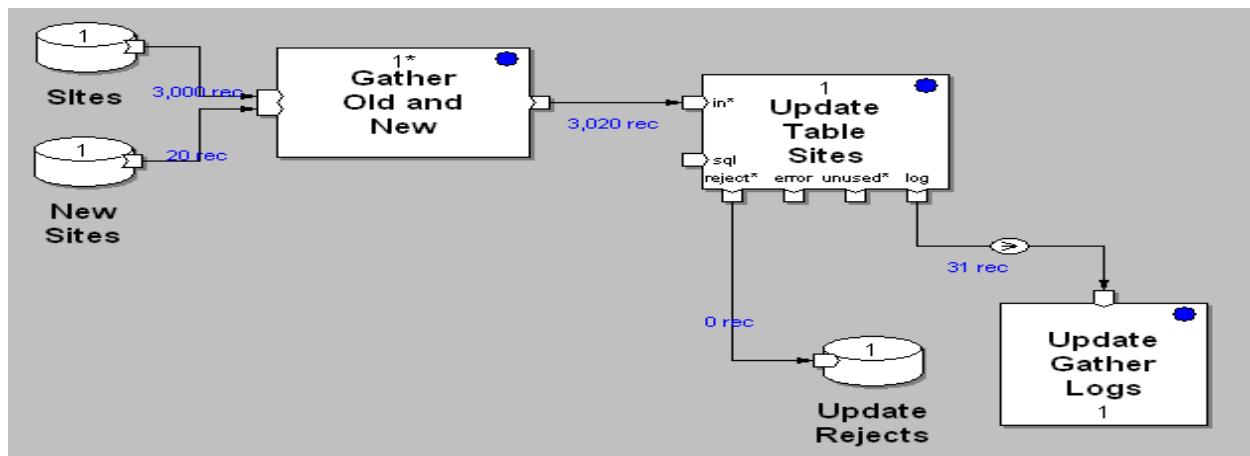
Connecting the log port provides information from the DBMS that might otherwise be “lost”.

Output file

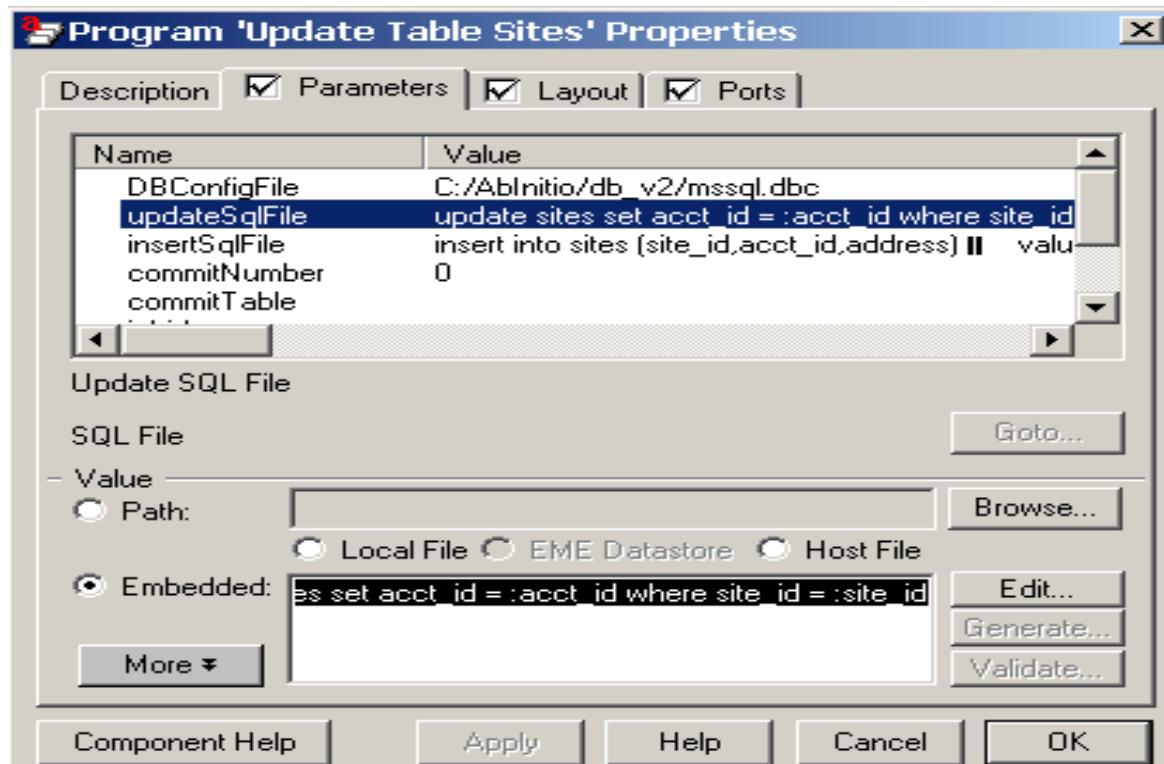


Update table

Update Table has two SQL statements. If statement 1 (the update in a classic “upsert”) fails then statement 2 (the insert in a classic “upsert”) gets executed.



Update time



Update table

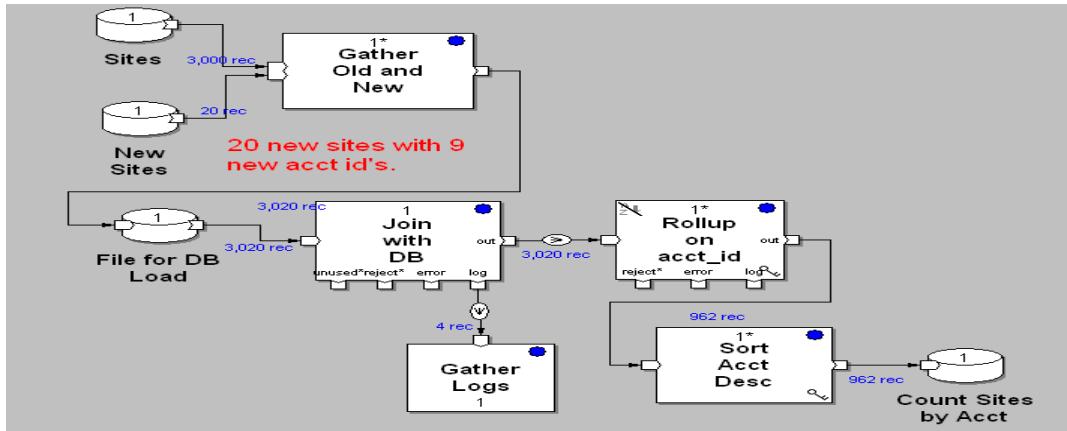
```
Parameter 'LogFile' of program 'Update Gather Logs' - log.log
File Edit Help
laptop-12.abinitio.com|Mon Dec 16 12:05:05 2002|Update_Gather_Logs.000||sta
laptop-12.abinitio.com|Mon Dec 16 12:05:05 2002|Update_Table_Sites.000|upda
laptop-12.abinitio.com|Mon Dec 16 12:05:07 2002|Update_Table_Sites.000|upda
|
laptop-12.abinitio.com|Mon Dec 16 12:05:07 2002|Update_Table_Sites.000|upda
    values(:site_id, :acct_id, :address);
|
laptop-12.abinitio.com|Mon Dec 16 12:05:33 2002|Update_Table_Sites.000|upda
    3020 records read
    3000 rows updated by SQL1
    20 records sent to SQL2
    20 rows updated by SQL2
    0 records rejected
    0 records unused|
laptop-12.abinitio.com|Mon Dec 16 12:05:33 2002|Update Table Sites.000|upda
```

Join with DB

Join with DB provides a mechanism for using an existing DBMS Table as the “lookup” for field(s) in the record stream.

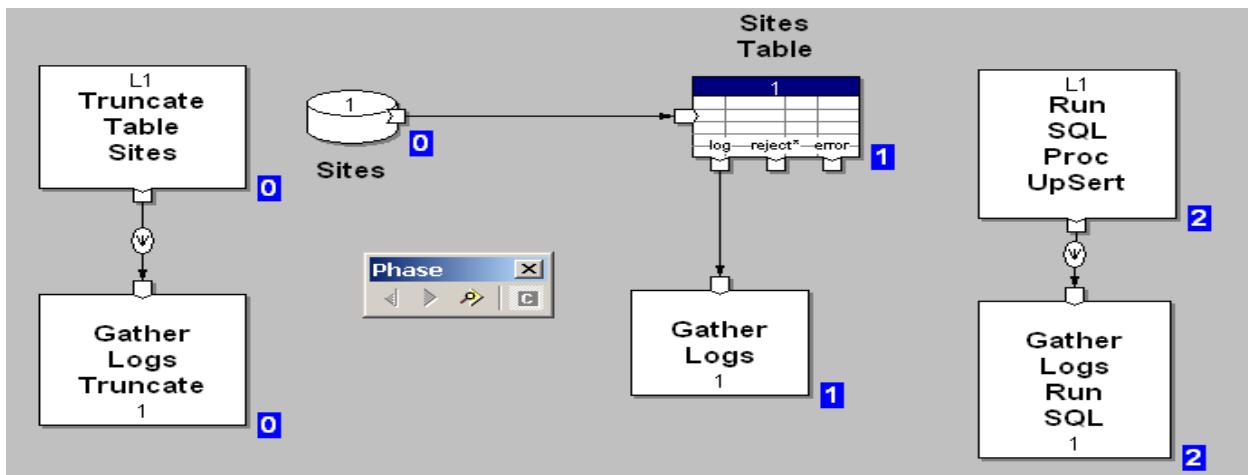
If “looking up” on a table with an index of the table it has the advantage over a Lookup File of capturing unused or of “inserting” a value that is not found into the table so subsequent matches can occur on that value.

- Join with DB joins records from the flow or flows connected to its in port with records read directly from a database, and outputs new records containing data based on, or calculated from, the joined records.



- The key consideration is that the Join with DB component executes a separate DB query per each input record. As a rule of thumb, whenever the number of input records is at least 10% of the table being referenced (i.e., a significant number of records), then you are better off unloading the data into flat files and using a join component (i.e., option 1). If, on the other hand, you are looking at less than 5% of the input records (i.e., a relatively small percentage of records within a large table), then using option 2 is likely to provide better performance.
- Basically, the expected performance of a database lookup or join will vary given the size of the table, and will depend on such things as the number of records that are being compared to the table (i.e., a large or small percentage of the records), etc.

Truncate table and run SQL



TRUNCATE TABLE

- Truncate Table deletes all rows in a database table, and writes confirmation messages to the log port.
- If you are truncating a table in order to load new data immediately afterward, rather than using a separate Truncate Table component, it is more efficient to set the Output Table truncate switch (on the Access properties tab).
- Parameters

DBConfigFile

Name of the database table configuration file.

Table

Name of the table to truncate.

RUN SQL

- Run SQL executes SQL statements in a database and writes confirmation messages to the log port. You can use Run SQL to perform database operations such as table or index creation.
- DBConfigFile

Name of the database table configuration file.

- SQLFile

Name of the file containing SQL statements to be executed against the database.

Note that the SQL statements, instead of being located in a file, can be embedded in the component.

M_DB UTILITY

`m_db command [options]` - Command line utility

- Performs various operations for working with databases. The operation performed is controlled by the Command and options, which follow the form of `m_db`.

`m_db gencfg -dbms_name > protodbc`

- will generate a skeleton database configuration file from the command line

`m_db create my_oracle_configdbc -dml_file test.dml -table test_table`

- will create an empty table whose record definition will be as defined in the DML file

Other possible values for command include gendml, genddl, load, unload, truncate, test ..etc

`m_db -help`

- For Help on `m_db` commands and syntax.

PERFORMANCE IMPROVEMENT IN GRAPH

1. Avoid Join with DB where ever possible as record processing is record-by-record.
2. As Update Table is a record-by-record processor it is better to use output table if possible.
3. Where ever required use table fields with index as key.
4. Use parallel load instead serial load for better performance, how ever both options have their own pro's and con's.

CASE STUDY

- Develop a Graph for the following scenario :
- We have a data warehouse built on Oracle database which has around 20 tables.

These tables' needs to be migrated to Teradata environment. The history data which is available in old oracle environment needs to be migrated to the new environment. Develop a generic graph for migrating data from Oracle database to Teradata environment.

HINTS:

- 1. Database/Schema name needs to be passed at runtime
- 2. Table name needs to be passed at runtime.
- 3. Dml have to be generated at run time based the input parameter –table name
- 4. Xfr's needs to be selected at runtime based on the table name.

Note: All the above mentioned should be passed at run time and should not be hard coded as this needs to be a generic graph for migrating all the tables.

CASE STUDY 2

- Develop a Graph for the following scenario :
- There is a database where you have a table called account_tbl which has the following attributes:

1. account_id number(10)

2. account_name varchar2(30)

3. location_id number(20)

- There is another location_tbl table in another database where you have location id to address mapping

1. location_id number(10)

2. addr_line_1 varchar2(30)

3. addr_line_2 varchar2(30)

4. addr_line_3 varchar2(30)

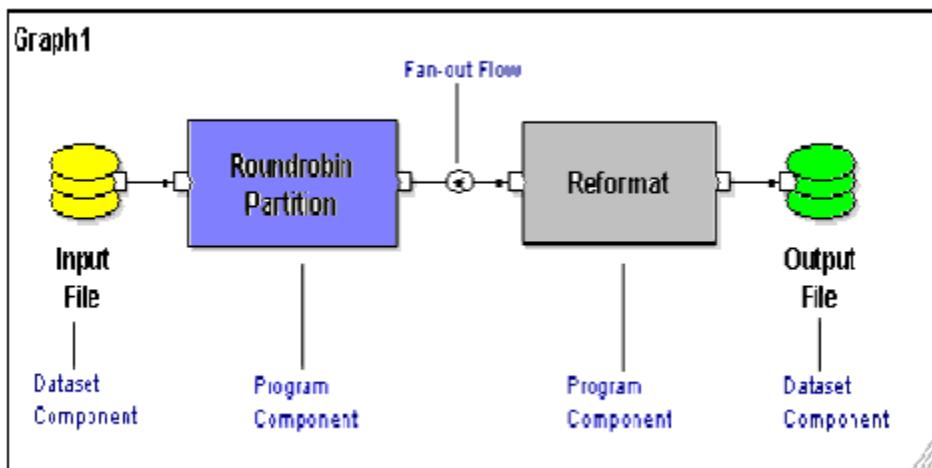
- we need to ftp to a site the account name and the address of the person in a compressed format.
- Scenario1 : Think the tables are huge like 10 million on account and location table of 4 million
- Scenario2 : Think the tables like 1 million on account and location table of 10000.

ABINITIO COMPONENTS HAVE A LOOK AT

COMPONENT GROUPS

- ◆ Compress components
- ◆ Continuous components
- ◆ Database components
- ◆ Dataset components
- ◆ Departition components
- ◆ Deprecated components
- ◆ FTP components
- ◆ Miscellaneous components
- ◆ Partition components
- ◆ Sort components
- ◆ Transform components
- ◆ Translate components
- ◆ Validate components

EXAMPLE FOR A GRAPH



FLOWs

- ◆ Flows connect components via ports.
- ◆ Four kinds of flows:
 - straight
 - fan-in
 - fan-out

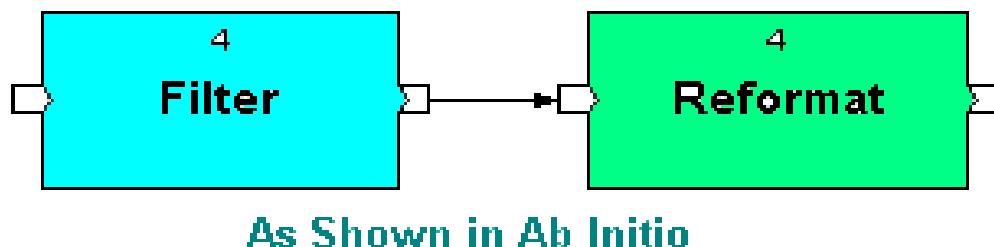
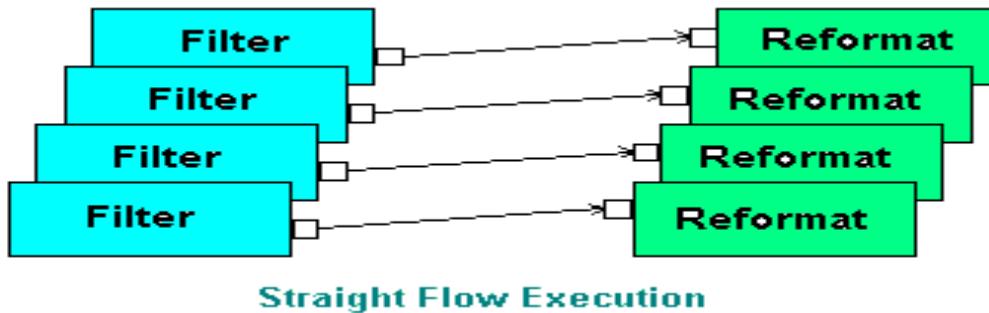
all-to-all

PORtS

- ◆ A port is a connection point for the input or output to a component.
- ◆ Every port has a record format.

STRAIGHT FLOW

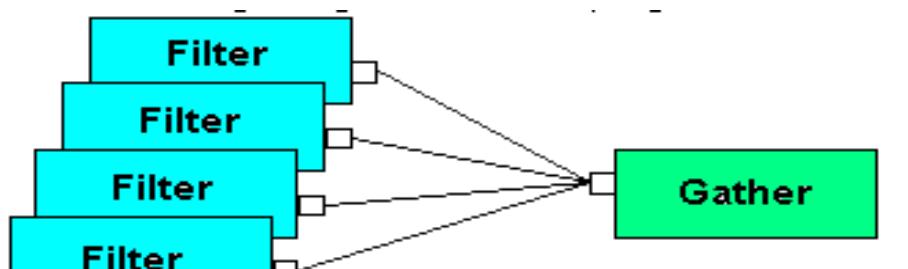
- ◆ Straight flows connect components that have the same number of partitions



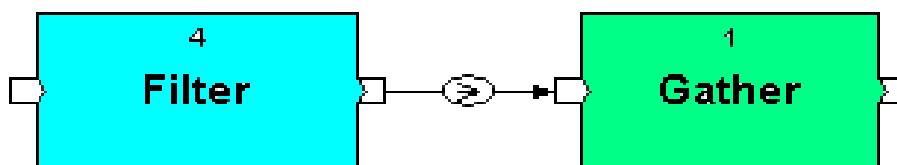
FAN – IN

- ◆ Fan-in flows connect components with a large number of partitions to components with a smaller number of partitions.
- ◆ The most common use of fan-in is to connect flows to Departition components.

ILLUSTRATION FOR FAN – IN



Fan-In Flow Execution

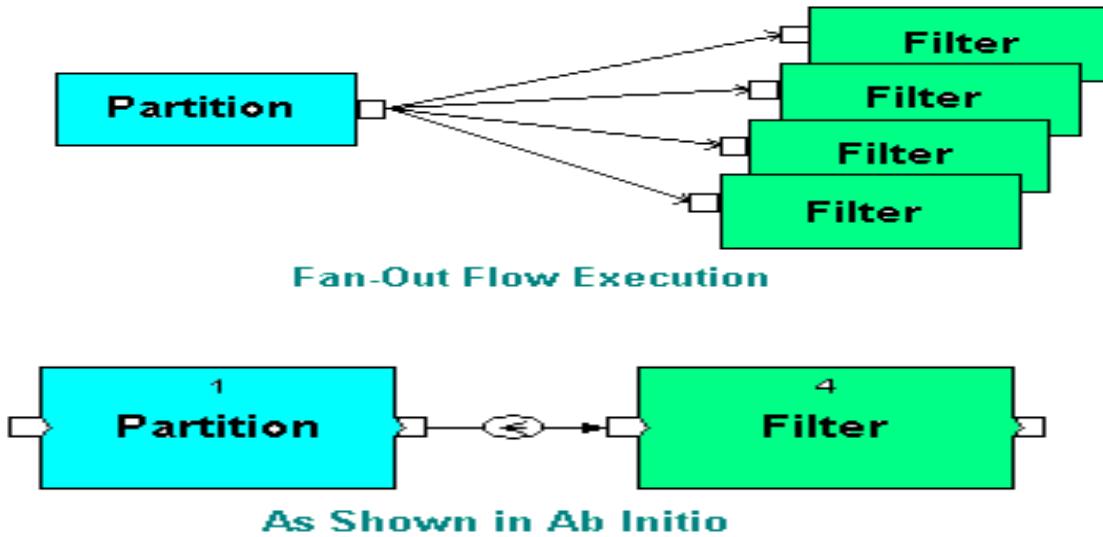


As Shown in Ab Initio

FAN-OUT

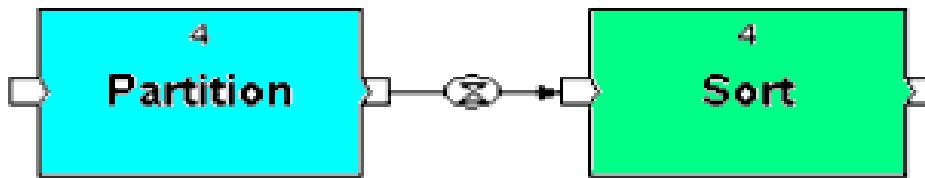
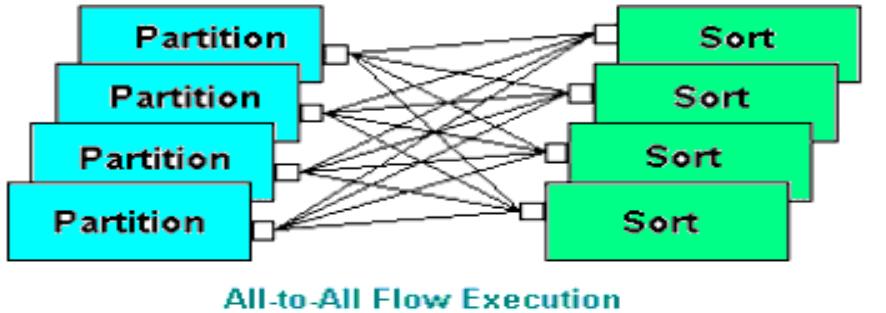
- ◆ Fan-out flows connect components with a small number of partitions to components with a larger number of partitions.
- ◆ The most common use of fan-out is to connect flows from partition components.
- ◆ This flow pattern is used to divide data into many segments for performance improvements.

FAN OUT FLOW



ALL-TO-ALL

- ◆ All-to-all flows typically connect components with different numbers of partitions.
- ◆ Data from any of the upstream partitions is sent to any of the downstream partitions.



As Shown in Ab Initio

COMPRESS COMPONENT

- ◆ Compress
- ◆ Uncompress
- ◆ GUnzip
- ◆ GZip

Compress reduces the volume of data in a flow.



Uncompress reverses the effects of Compress.

GUNZIP COMPONENT

GUnzip restores compressed data, reversing the effect of the Compress or GZip components

GZIP COMPONENT

GZip reduces the volume of data in a flow.

PARTITION COMPONENTS

PURPOSE : Partition components distribute data records to multiple flow partitions to support data parallelism, as follows

- ◆ Broadcast
- ◆ Partition by Expression
- ◆ Partition by Key
- ◆ Partition by Percentage
- ◆ Partition by Range
- ◆ Partition by Round-robin
- ◆ Partition with Load Balance

BROADCAST

PURPOSE : Broadcast arbitrarily combines all the data records it receives into a single flow and writes a copy of that flow to each of its output flow partitions..

DETAILS :

- ◆ Broadcast does not support Default record assignment .

THE BROADCAST COMPONENT:

- ◆ Reads records from all flows on the in port
- ◆ Combines the records arbitrarily into a single flow
Copies all the records to all the flow partitions connected to the out port
- ◆ Use Broadcast to increase data parallelism when you have connected a single fan-out flow to the out port or to increase component parallelism when you have connected multiple straight flows to the out port.
- ◆ Copies all the records to all the flow partitions connected to the out port
- ◆ Use Broadcast to increase data parallelism when you have connected a single fan-out flow to the out port or to increase component parallelism when you have connected multiple straight flows to the out port.

PARTITION BY EXPRESSION.

PURPOSE: Partition by Expression distributes data records to its output flow partitions according to a specified DML expression.

DETAILS

Partition by Expression accepts fan-out flows on its out ports

THE PARTITION BY EXPRESSION COMPONENT

- ◆ Reads records in arbitrary order from the flows connected to the in port
- ◆ Distributes the records to the flows connected to the out port, according to the expression in the function parameter

PARTITION BY KEY

PURPOSE : Partition by Key distributes data records to its output flow partitions according to key values.

DETAIL

- ◆ Partition by Key is typically followed by Sort

The Partition by Key component:

- ◆ Reads records in arbitrary order from the in port
- ◆ Distributes them to the flows connected to the out port, according to the key parameter, writing records with the same key value to the same output flow

PARTITION BY PERCENTAGE

PURPOSE : Partition by Percentage distributes a specified percentage of the total number of input data records to each output flow .

DETAIL :

- ◆ Reads records from the in port
- ◆ Writes a specified percentage of the input records to each flow on the out port

We can supply the percentages in either of two ways:

- ◆ By specifying the percentages in the percentages parameter.
- ◆ By connecting the output of any component that produces a list of percentages to the pct port of Partition by Percentage. Use decimal('\n') as the record format for the pct port of Partition by Percentage.

PARTITION BY RANGE

PURPOSE : Partition by Range distributes data records to its output flow partitions according to the ranges of key values specified for each partition.

DETAIL : Typically, you route the output from the out port of Find Splitters to the split port of Partition by Range. When you do this, also do the following:

- ◆ Use the same key specifier for both components.

The Partition by Range component:

- ◆ Reads splitter records from the split port, and assumes that these records are sorted according to the key parameter.
- ◆ Determines whether the number of flows connected to the out port is equal to n (where n-1 represents the number of splitter records).

- ◆ If not, Partition by Range writes an error message and stops the execution of the graph.
- ◆ Reads data records from the flows connected to the in port in arbitrary order.
- ◆ Distributes the data records to the flows connected to the out port according to the values of the key field(s), as follows:
 - ◆ Assigns records with key values less than or equal to the first splitter record to the first output flow.
 - ◆ Assigns records with key values greater than the first splitter record, but less than or equal to the second splitter record to the second output flow, and so on

PARTITION BY ROUND-ROBIN

PURPOSE : Partition by Round-robin distributes blocks of data records evenly to each output flow in round-robin fashion.

The Partition by Round-robin component:

- ◆ Reads records from the in port.
- ◆ Distributes them in blocksize chunks to its output flows according to the order in which the flows are connected

PARTITION WITH LOAD BALANCE

PURPOSE : Partition with Load Balance distributes data records to its output flow partitions, writing more records to the flow partitions that consume records faster .

The Partition with Load Balance component:

- ◆ Reads records in arbitrary order from the flows connected to its in port
- ◆ Distributes those records among the flows connected to its out port, sending more records to the flows that consume records faster
- ◆ Partition with Load Balance writes data records until each flow's output buffer fills up.

DEPARTITION COMPONENT

Departition components combine multiple flow partitions of data records into a single flow as follows:

- ◆ Concatenate
- ◆ Gather
- ◆ Interleave
- ◆ Merge

CONCATENATE

PURPOSE : Concatenate appends multiple flow partitions of data records one after another.

The Concatenate component:

- ◆ Reads all the data records from the first flow connected to the in port (counting from top to bottom on the graph) and copies them to the out port.
- ◆ Then reads all the data records from the second flow connected to the in port and appends them to those of the first flow, and so on.

GATHER

PURPOSE : Gather combines data records from multiple flow partitions arbitrarily.

DETAIL :

We can use Gather to:

- ◆ Reduce data parallelism, by connecting a single fan-in flow to the in port
- ◆ Reduce component parallelism, by connecting multiple straight flows to the in port
- ◆ Reads data records from the flows connected to the in port
- ◆ Combines the records arbitrarily
- ◆ Writes the combined records to the out port

INTERLEAVE

PURPOSE : Interleave combines blocks of data records from multiple flow partitions in round-robin fashion.

DETAIL :

- ◆ Interleave reads from its flows in a specific order, so it may cause deadlock

The Interleave component:

- ◆ Reads the number of data records specified in the blocksize parameter from the first flow connected to the in port.
- ◆ Then reads the number of data records specified in the blocksize parameter from the next flow, and so on .
- ◆ Writes the records to the out port

MERGE

PURPOSE : Merge combines data records from multiple flow partitions that have been sorted according to the same key specifier, and maintains the sort order.

DATASET COMPONENTS

- ◆ Input File
- ◆ Input Table
- ◆ Intermediate File
- ◆ Join with DB
- ◆ Lookup File
- ◆ Output File
- ◆ Output Table
- ◆ SAS Input File
- ◆ SAS Output File

INPUT FILE

- ◆ It represents data records read as input to a graph from one or multiple serial files or from a multifile
- ◆ It unloads data records from a database into an Ab Initio graph, allowing you to specify as the source either a database table, or an SQL statement that selects data records from one or more tables.
- ◆ Input Table is located in both the Database and Dataset folders in the Component Organizer of the GDE.

INTERMEDIATE FILE

- ◆ The upstream component writes to Intermediate File through Intermediate File's write port. After the flow of data records into the write port is complete, the downstream component reads from Intermediate File's read port. This guarantees that the writing and reading processes are in two separate phases.

JOIN WITH DB

- ◆ It joins records from the flow or flows connected to its input port with records read directly from a database, and outputs new records containing data based on, or calculated from, the joined records.

LOOKUP FILE

- ◆ Unlike other datasets, you do not connect Lookup Files to other components.
- ◆ Although a Lookup File appears in the graph without any connected flows, its contents are accessible from other components in the same or later phases.
- ◆ You use the Lookup File in other components by calling one of the DML functions `lookup` , `lookup_count` , or `lookup_next` in any transform function or expression parameter.

OUTPUT TABLE

It loads data records from a graph into a database, letting you specify the records' destination either directly as a single database table, or through an SQL statement that inserts data records into one or more tables.

OUTPUT FILE

- ◆ It represents data records written as output from a graph into one or multiple serial files or a multifile.

- ◆ **SAS Input File:** It represents data records read as input to a graph from a SAS dataset.
- ◆ **SAS Output File :** It represents data records written as output from a graph into a SAS dataset

TRANSFORM COMPONENTS

"One or several transform functions are used to Modify or Manipulate data records"

COMMON TO ALL TRANSFORM COMPONENTS

Ports : -Gives input/out behavior of the program

- in
- out
- reject
- error
- log

- ◆ **Parameters:** - It specifies some aspect of graph, subgraph, or component behavior.

- key
- limit
- ramp
- reject threshold
- select
- sorted input

- a. Input must be sorted or grouped
- b. Input need not be sorted (In memory).

TRANSFORM FUNCTIONS

- ◆ Aggregate

- ◆ Dedup Sorted
- ◆ Denormalize Sorted
- ◆ Filter by Expression
- ◆ Join
- ◆ Normalize
- ◆ Reformat
- ◆ Rollup
- ◆ Scan

Aggregate :- Generates data records that summarize groups of data records.

Dedup Sorted : -Separates one specified data record in each group of data records from the rest of the records in the group.

Denormalize Sorted:-Consolidates groups of related data records into a single data record with a vector field for each group

- ◆ **Filter by Expression:** - Filters data records according to a specified DML expression.
- ◆ **Join** :-performs inner,outer, and semi-joins with flows of data records.
- ◆ **Normalize:** -Generates multiple output data records from each input data record.The number of records depend on a field or field s in each records
- ◆ **Reformat:** -It changes the record format of data records
- ◆ **Rollup:** -It generates data records that summarize groups of data records.
- ◆ **Scan:** - Generates a series of cumulative summary records.

<SLIDE 9 & 10 > REFERRED MORE ON UNIX & SHELL SCRIPT

UNIX OP SYS.

- An *operating system* (or "OS") is a set of programs that controls a computer
- It controls both the

- *hardware* (things you can touch—such as keyboards, displays, and disk drives)
- *software* (application programs that you run, such as a word processor).
- Some computers have a *single-user* OS, which means only one person can use the computer at a time. They can also do only one job at a time.

But if it has a *multiuser, multitasking* operating system like UNIX. Then these powerful OSes can let many people use the computer at the same time and let each user run several jobs at once

VERSIONS

- Now there are many different versions of UNIX.
 - At first there were two main versions:
 - The line of UNIX releases that started at AT&T (the latest is System V Release 4),
 - And from the University of California at Berkeley (the latest version is BSD 4.4).
 - Now commercial versions include SunOS, Solaris, SCO UNIX, SG IRIX, AIX, HP/UX
 - The freely available versions include Linux and FreeBSD 5.2 (based on 4.4BSD)
 - Many Versions of Linux - Redhat, Fedroa, Debian, SuSE and MandrakeSoft
 - Apple Mac OS X (FreeBSD 5.2)
-

KERNEL

- Manages memory and allocates it to each process
- Schedules work done by the CPU
- Organizes transfer of data from one part of machine to another

- Accepts instructions from shell and carries them out
- Enforces access permission on the file system

SHELL

- Command interpreter
- Create customized environments
- Write shell scripts
- Define command aliases
- Manipulate command history
- File and command completion
- Edit the command line

UNIX SYSTEM PROGRAMMING

- Programs make system calls (also called supervisor calls to invoke kernel).
- A system call is essentially a procedure call into the operating system
 - The procedure call is protected
- Types of system calls
 - File I/O
 - Process management
 - Inter-process communication (IPC)
 - Signal handling

SYSTEM CALLS

VARIOUS UNIX SHELLS

- sh (Bourne shell)
- ksh (Korn shell)
- csh (C shell)
- tcsh
- bash
- Differences mostly in scripting details
- Two main flavors of Unix Shells
 - Bourne (or Standard Shell): sh, ksh, bash, zsh
 - Fast
 - \$ for command prompt
 - C shell : csh, tcsh
 - better for user customization and scripting
 - %, > for command prompt
- To check shell:
 - % echo \$SHELL (shell is a pre-defined variable)
- To switch shell:
 - % exec shellname (e.g., % exec bash)

KORN SHELL

- We will be using ksh as the standard shell for examples in this class
- Language is a superset of the Bourne shell (sh)

MAJOR FEATURES

Job control
Additional built-in variables
Additional pattern matching
Additional options for test
Typeset
Read
Parameter Expansion

LOGIN SCRIPTS

- You don't want to enter aliases, set environment variables, set up command line editing, etc. each time you log in
- All of these things can be done in a script that is run each time the shell is started
- For ksh:
 - `~/.profile` - is read for a login shell
 - `~/.kshrc`
- For tcsh
 - `~/.login`
 - `~/.cshrc`

EXAMPLE .PROFILE [PARTIAL]

```
# set ENV to a file invoked each time sh is started for interactive use.  
  
ENV=$HOME/.shrc; export ENV  
  
HOSTNAME=`hostname` ; export HOSTNAME  
  
PS1="$USER@$HOSTNAME>"  
  
alias 'll'='ls -l'
```

```
alias 'la'='ls -la'  
alias 'ls'='ls -F'  
alias 'rm'='rm -i'  
alias 'm'='more'  
set -o vi  
echo ".profile was read"
```

STDOUT STDIN STDERR

- Each shell (and in fact all programs) automatically open three “files” when they start up
 - Standard input (stdin): Usually from the keyboard
 - Standard output (stdout): Usually to the terminal
 - Standard error (stderr): Usually to the terminal
 - Programs use these three files when reading (e.g. cin), writing (e.g. cout), or reporting errors/diagnostics

REDIRECTING STDOUT

- Instead of writing to the terminal, you can tell a program to print its output to another file using the > operator
- >> operator is used to append to a file
- Examples:
 - man ls > ls_help.txt
 - Echo \$PWD > current_directory
 - cat file1 >> file2

REDIRECTING STDERR

- Instead of writing errors to the terminal, you can tell a program to write them to another file using the:
 - ksh: 2> operator
 - tcsh: >& operator
 - Examples (suppose j is a file that does not exist)

```
crdap205:/home/cchome01/rk45694 -> cat IPL_team.list
```

```
cat: cannot open IPL_team.list
```

```
crdap205:/home/cchome01/rk45694 -> cat IPL_team.list 2> error_file.txt
```

```
crdap205:/home/cchome01/rk45694 -> cat error_file.txt
```

```
cat: cannot open IPL_team.list
```

REDIRECTING STDIN

- Instead of reading from the terminal, you can tell a program to read from another file using the < operator
- Examples:
 - Mail user@domain.com < message
 - interactive_program < command_list

PIPE AND FILTERS

- **Pipe:** a way to send the output of one command to the input of another
- **Filter:** a program that takes input and transforms it in some way
 - wc - gives a count of words/lines/chars
 - grep - searches for lines with a given string
 - more
 - sort - sorts lines alphabetically or numerically

EXAMPLES OF FILTERING

- ls -la | more
- cat file | wc
- man ksh | grep "history"
- ls -l | grep "dkl" | wc
- who | sort > current_users

UNIX FILE SYSTEM

- The filesystem is your interface to
 - physical storage (disks) on your machine
 - storage on other machines
 - output devices
 - etc.
- Everything in UNIX is a file (programs, text, peripheral devices, terminals, ...) and every file has a name
- Case sensitive
- Unix file names can contain any characters (although some make it difficult to access the file)
- Unix file names can be long
- There are no drive letters in UNIX! The file system provides a logical view of the storage devices

WORKING DIRECTORY

- The current directory in which you are working
- pwd command: outputs the absolute path (more on this later) of your working directory
- Unless you specify another directory, commands will assume you want to operate on the working directory

HOME DIRECTORY

- A special place for each user to store personal files
 - When you log in, your working directory will be set to your home directory
 - Your home directory is represented by the symbol ~ (tilde)
 - The home directory of "user1" is represented by ~user1
-
-

Path names

Absolute Path name

* A file is identified by the path name from the root

e.g.,

/usr/trg/c/test.c

where

- *test.c* is an ordinary file
- *usr, trg, c* are directories
- *trg* is a sub-directory under *usr*

Relative path name

* UNIX keeps track of the user's current directory

* If a "/" does not precede a file name then the name interpretation begins with the current directory

e.g.,

If current directory is

/usr/trg

then the file could be just referenced as *c/test.c*

WHAT GOES WHERE

- /
 - Root of the entire system
 - Comparable in Windows to C: (but you may have D:, E: .. several roots)
- /bin
 - Commonly used binaries (programs)
- /sbin
 - More programs to run
- /usr
 - User related commands as well as a whole bunch of random stuff
- /lib
 - Libraries go in here
- /dev
 - All devices are located in here
- /home
 - Traditionally, this is where user accounts are stored
- /etc
 - Startup files and configuration files for daemons and other programs
- /var
 - The /var directory contains files that change as the system is running.
This includes:
 - /var/log
 - Directory that contains log files. These are updated as the system runs. You should view the files in this directory from time to time, to monitor the health of your system.

- Traditional location of mailboxes
 - /var/spool/mail
 - /proc
- Special files that contain information about the system or info from running programs

TYPES OF FILES

- Plain (- in the first bit)
 - Most files
 - Includes binary and text files
- Directory (d)
 - A directory is actually a file
 - Points to another set of files
- Link (l): A pointer to another file or directory
- Special: e.g. peripheral devices

PROGRAM AND PROCESS

- Program is an executable file that resides on the disk.
- Process is an executing instance of a program.
- A Unix process is identified by a unique non-negative integer called the process ID.
- Check process status using the “ps” command.
 - After a successful login, the shell program is run.
 - /home/userid\$ ps

PID TTY TIME CMD

23110 pts/3 0:00 ps

18666 pts/3 0:00 ksh

FG AND BG PROCESS

- A program run using the ampersand operator “&” creates a background process.
E.g.: /home/userid\$ back &
- otherwise it creates a foreground process.
E.g.: /home/userid\$ back
- Only 1 foreground process for each session. Multiple background processes.
- Where are background processes used?
- All system daemons, long user processes, etc.
e.g. printer-daemon process or mailer-daemon process.
- These processes are always running in background.

TO STOP A PROCESS

- Foreground processes can generally be stopped by pressing CONTROL C (^C).
- Background processes can be stopped using the kill command.
- Usage: kill SIGNAL <process id list>
- kill -9 <process id list> (-9 means no blocked) Or kill <process id list>.
- If a foreground process is not stopping by ^C, you can open another session and use the kill command.

169.172.51.17 - PuTTY

```
login as: rkuppus
rkuppus's Password:
*****
* 
* *** Welcome to CAM Production NODES ***
* 
* You are authorized to use this System for approved business purposes only.
* Use for any other purpose is prohibited. All transactional records,
* reports, e-mail software, and other data generated by or residing upon
* this System are the property of the Company and may be used by the Company
* for any purpose. Authorized and unauthorized activities may be monitored.
* 
* ****
[YOU HAVE NEW MAIL]

[rkuppus@campdb0602: /u/rkuppus]
```

169.172.51.17 - PuTTY

```
rkuppus's Password:
*****
* 
* *** Welcome to CAM Production NODES ***
* 
* You are authorized to use this System for approved business purposes only.
* Use for any other purpose is prohibited. All transactional records,
* reports, e-mail software, and other data generated by or residing upon
* this System are the property of the Company and may be used by the Company
* for any purpose. Authorized and unauthorized activities may be monitored.
* 
* ****
[YOU HAVE NEW MAIL]

[rkuppus@campdb0602: /u/rkuppus]
$ ls
total 65721
drwxr-xr-x 11 rkuppus camuser      512 Nov 23 22:08 .
dr-xr-xr-x  9 root      system       9 Nov 27 06:51 ..
-rwxrwxr--  1 rkuppus camuser     4530 Jun 23 04:41 .env*
-rw-r--r--  1 rkuppus camuser    1904 May 11 2005 .profile
-rw-r--r--  1 rkuppus camuser     10 May 11 2005 .rhosts
-rw-------  1 rkuppus camuser   12924 Sep 28 09:32 .sh_history
drwxr--S--- 2 rkuppus camuser      512 Oct 19 14:01 .ssh/
drwxrwsrwx  3 rkuppus camuser      512 Jun 01 02:15 abworkarea/
-rw-rw-rw-  1 rkuppus camuser  33559207 Aug 20 22:07 core
drwxrwsrwx  3 rkuppus camuser      512 Nov 23 22:19 data1/
drwxrwsrwx  3 rkuppus camuser      512 Jun 01 01:12 dblxxx/
drwxrwsrwx  3 rkuppus camuser      512 Jun 01 02:40 dev/
-rwxrwxrwx  1 rkuppus camuser      0 May 11 2005 nohup.out*
-rwxrwxrwx  1 rkuppus camuser      4 May 18 2005 out*
drwxrwsrwx  3 rkuppus camuser      512 Jun 01 01:18 prodrun/
drwxrwsrwx  2 rkuppus camuser      512 Sep 08 01:17 ram/
drwxrwsrwx  5 rkuppus camuser  2048 Nov 27 05:59 ramesh/
-rwxrwxrwx  1 rkuppus camuser    222 May 18 2005 smit.log*
-rwxrwxrwx  1 rkuppus camuser    0 May 18 2005 smit.script*
-rwxrwxrwx  1 rkuppus camuser    0 May 18 2005 smit.transaction*
drwxrwsrwx  2 rkuppus camuser      512 Nov 22 08:08 sree/
-rwxrwxrwx  1 rkuppus camuser   2523 May 18 2005 user.file*
```

169.172.51.17 - PuTTY

```
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/india]
$ pwd
/u/rkuppus/ramesh/sachin/india      Absolute path

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/india]
$ cd /u/rkuppus/ramesh

[rkuppus@campdb0602: /u/rkuppus/ramesh]
$ pwd
/u/rkuppus/ramesh

[rkuppus@campdb0602: /u/rkuppus/ramesh]
$ Always has the '/' symbol
```



The screenshot shows a Windows desktop environment. In the foreground, a PuTTY terminal window is open, displaying a Linux command-line session. The session starts at the root directory and demonstrates navigating through subdirectories. The terminal window has a blue header bar with the text "169.172.51.17 - PuTTY". The main area of the terminal shows the following commands and output:

```
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/india]
$ pwd
/u/rkuppus/ramesh/sachin/india          Relative path

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/india]
$ cd ..
with reference to the current directory

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cd .../..

[rkuppus@campdb0602: /u/rkuppus]
$ cd ramesh

[rkuppus@campdb0602: /u/rkuppus/ramesh]
$
```

The taskbar at the bottom of the screen shows several open applications: "start", "169.172.51.17 - PuTTY" (which is currently active), "Microsoft PowerPoint...", and "s1 - Paint". The system tray indicates the date and time as "11:36 AM".

Features

- * Hierarchical
- * Security on each file
 - Owner
 - Group
 - All others
- * Separate security for
 - read
 - write and
 - execute

- * Removable
- * File Independence -
- * Time stamp on each file
 - Modification time
 - Access time

crdap205:/home/cchome01/rk45694/sample_dir -> ls -lrt

total 16

```
drwxr-xr-x  2 rk45694  devlgrp    96 Jun  3 03:52 sample1_dir/
drwxr-xr-x  2 rk45694  devlgrp    96 Jun  3 03:52 sample2_dir/
-rwxr--r--  1 rk45694  devlgrp   34 Jun  3 04:01 welcome.ksh*
```

- Symbol in the position 0 ("d") is the type of the file.

It is either "d" if the item is a directory, or "l" if it is a link, or "-" if the item is a regular file.

- Symbols in positions 1 to 3 ("rwx") are permissions for the owner of the file.
- Symbols in positions 4 to 6 ("r--") are permissions for the group.
- Symbols in positions 7 to 9 ("r--") are permissions for others.

/usr/games - Game Programs

/usr/include - Include files for language procedure

Examples : C-header files

stdio.h, math.h

/usr/lib - Archive libraries

Example : *troff*

/usr/mail - Mail files

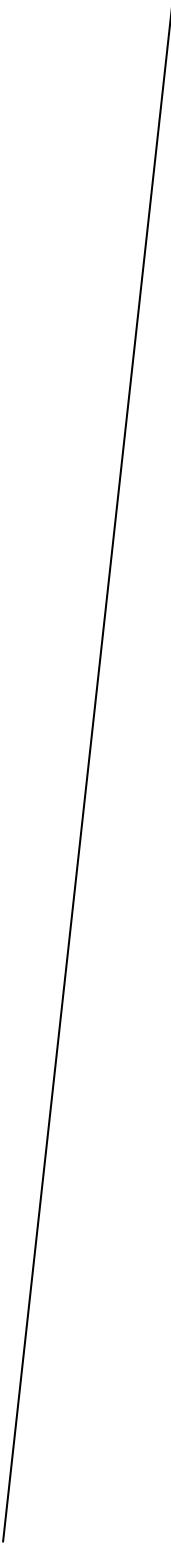
Example : *mailbox*

/usr/news - News files

/usr/spool - Spool files

/usr/tmp - Temporary files

/usr/src - Program Source Files



```

169.172.51.17 - PuTTY

-rw-rw-rw- 1 rkuppus camuser 0 Nov 27 07:14 football
-rw-rw-rw- 1 rkuppus camuser 0 Nov 27 07:15 basketball
-rw-rw-rw- 1 rkuppus camuser 0 Nov 27 07:15 boxing
-rw-rw-rw- 1 rkuppus camuser 0 Nov 27 07:15 tabletennis
-rw-rw-rw- 1 rkuppus camuser 334 Nov 27 07:19 list1
-rw-rw-rw- 1 rkuppus camuser 387 Nov 27 07:19 liat2
-rw-rw-rw- 1 rkuppus camuser 66 Nov 28 00:48 englandteam
drwxrwsrwx 2 rkuppus camuser 512 Nov 28 00:48 .

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ chmod 222 englandteam

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls -lrt
total 56
drwxrwsrwx 5 rkuppus camuser 2048 Nov 27 05:59 ../
--w--w--w- 1 rkuppus camuser 41 Nov 27 06:20 oldteam
-rw-rw-rw- 1 rkuppus camuser 66 Nov 27 07:12 newteam
-rw-rw-rw- 1 rkuppus camuser 0 Nov 27 07:14 football
-rw-rw-rw- 1 rkuppus camuser 0 Nov 27 07:15 basketball
-rw-rw-rw- 1 rkuppus camuser 0 Nov 27 07:15 boxing
-rw-rw-rw- 1 rkuppus camuser 0 Nov 27 07:15 tabletennis
-rw-rw-rw- 1 rkuppus camuser 334 Nov 27 07:19 list1
-rw-rw-rw- 1 rkuppus camuser 387 Nov 27 07:19 liat2
--w--w--w- 1 rkuppus camuser 66 Nov 28 00:48 englandteam
drwxrwsrwx 2 rkuppus camuser 512 Nov 28 00:48 .

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cp englandteam scot_team
cp: englandteam: The file access permissions do not allow the specified action.

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ 

```

Starting And Ending A Session

login : User can type his name and password to identify himself

login command can be used as

```
$ exec login
```

to log-on onto another user account after identifying yourself in response to prompts for user name and password

su setuser

This is used to become another user or super-user provided the password is known.

e.g., \$su

Prompt the user for the superuser password \$su - trg2

Prompt the user for the password of user trg2 \$su - trg2 -c "ls -l"

Temporarily changes to trg2 and executes the command ls -l and comes back to the original user

passwd *Change the passwd for the user*

e.g., \$ passwd

Prompt you for old passwd and new passwd

logout *This command exits or logs-out from the current user and executes the file .logout before coming out*

e.g., \$ logout

or

\$ exit

or

\$ <ctrl-d>

exits from the current login

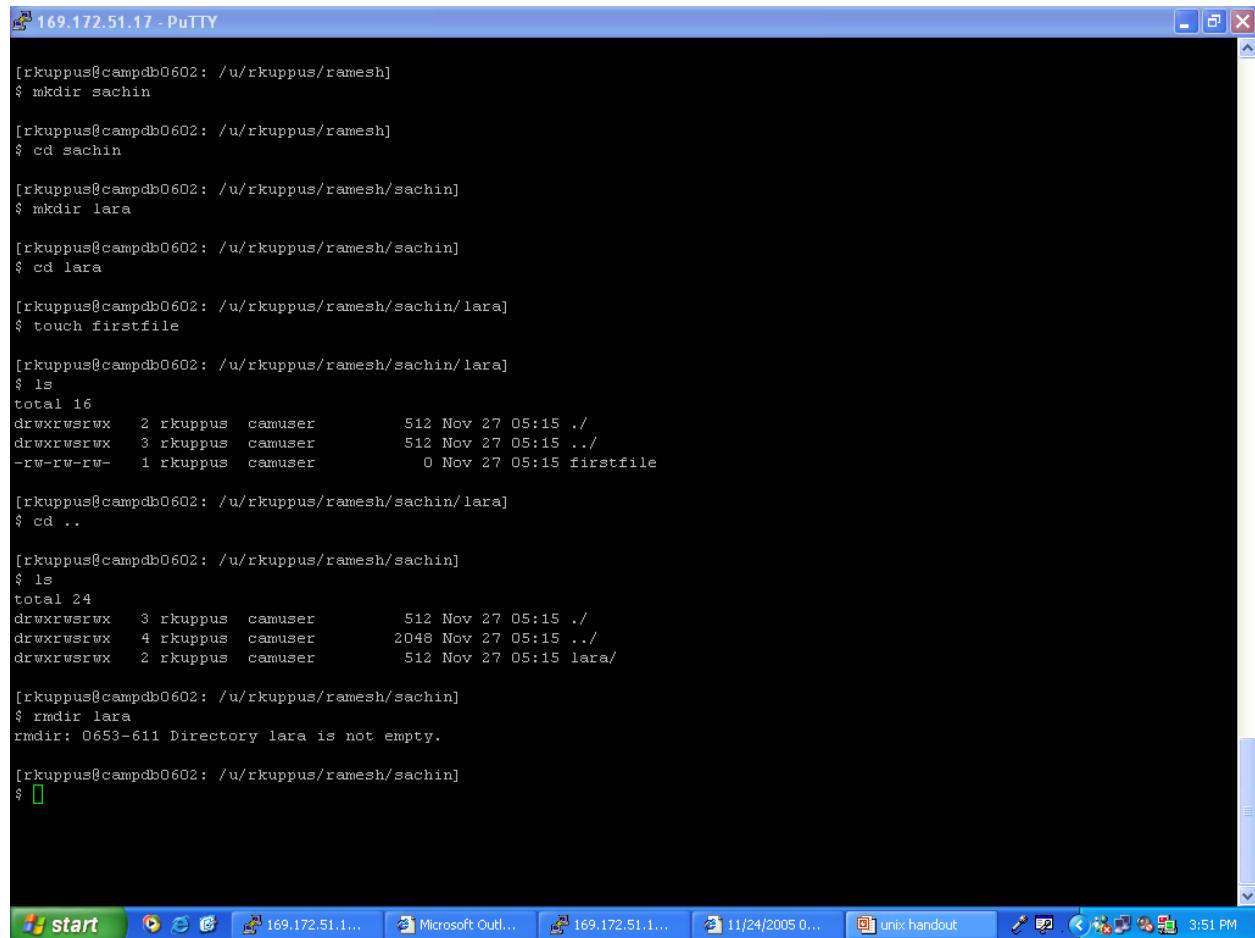
File Management Utilities

Directory Security Management	Operation	File Comp.	
		<i>cmp</i>	<i>passwd</i>
<i>cd</i>		<i>comm</i>	<i>crypt</i>
<i>pwd</i>			<i>chown</i>
<i>mkdir</i>			<i>chgrp</i>
<i>rmdir</i>			<i>umask</i>
<i>mvdir</i>			<i>chmod</i>
File contents	File compression	Mountable file	Copy, Move Remove & Time
<i>cat</i>	<i>pack</i>	<i>mount</i>	<i>cp</i>
<i>ls</i>	<i>unpack</i>	<i>umount</i>	<i>ln</i>
<i>wc</i>			<i>mv</i>
<i>file</i>			<i>rm</i>

FILE MANAGEMENT UTILITES

- mkdir* creates a new directory
- rm* removes a file
- rmdir* removes a directory
- du* displays disk usage
- df* displays number of free block
- touch* updates the time of last modification
- find* locates files that match certain area
- file* displays the type of file

pwd displays full pathname of current directory



The screenshot shows a Windows desktop environment with a PuTTY terminal window open. The terminal window title is "169.172.51.17 - PuTTY". The command history in the terminal is as follows:

```
[rkuppus@campdb0602: /u/rkuppus/ramesh]
$ mkdir sachin

[rkuppus@campdb0602: /u/rkuppus/ramesh]
$ cd sachin

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ mkdir lara

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cd lara

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/lara]
$ touch firstfile

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/lara]
$ ls
total 16
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 05:15 .
drwxrwsrwx  3 rkuppus  camuser      512 Nov 27 05:15 ..
-rw-rw-rw-  1 rkuppus  camuser       0 Nov 27 05:15 firstfile

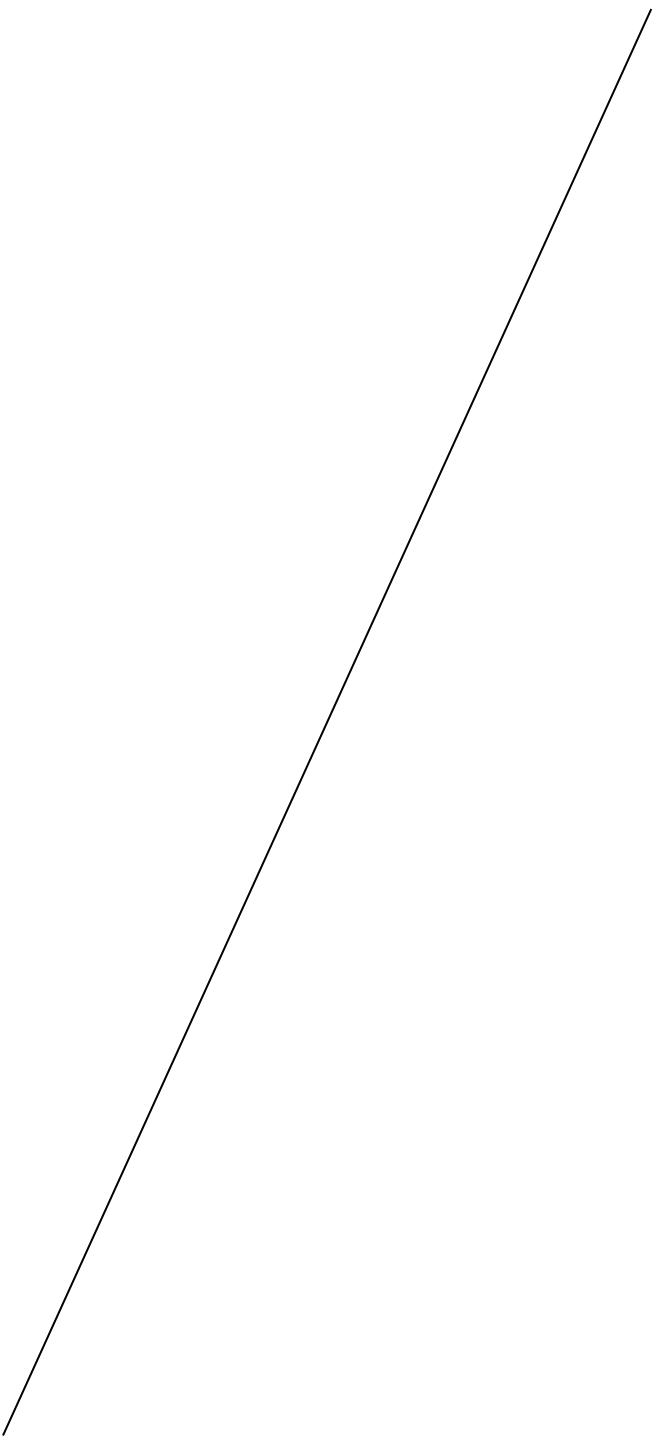
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/lara]
$ cd ..

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 24
drwxrwsrwx  3 rkuppus  camuser      512 Nov 27 05:15 .
drwxrwsrwx  4 rkuppus  camuser     2048 Nov 27 05:15 ..
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 05:15 lara/

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ rmdir lara
rmdir: 0653-611 Directory lara is not empty.

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ 
```

The taskbar at the bottom of the screen shows several icons, including Microsoft Outlook, Microsoft Internet Explorer, and a file named "unix handout". The system tray shows the date and time as "11/24/2005 0..." and "3:51 PM".



169.172.51.17 - PuTTY

```
drwxrwsrwx  4 rkuppus  camuser        2048 Nov 27 05:15 ../
drwxrwsrwx  2 rkuppus  camuser         512 Nov 27 05:15 lara/

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ rmdir lara
rmdir: 0653-611 Directory lara is not empty.

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cd lara

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/lara]
$ ls
total 16
drwxrwsrwx  2 rkuppus  camuser        512 Nov 27 05:15 .
drwxrwsrwx  3 rkuppus  camuser        512 Nov 27 05:15 ../
-rw-rw-rw-  1 rkuppus  camuser          0 Nov 27 05:15 firstfile

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/lara]
$ rm firstfile

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/lara]
$ ls
total 16
drwxrwsrwx  2 rkuppus  camuser        512 Nov 27 05:22 .
drwxrwsrwx  3 rkuppus  camuser        512 Nov 27 05:15 ../

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin/lara]
$ cd ..

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ rmdir lara

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 16
drwxrwsrwx  2 rkuppus  camuser        512 Nov 27 05:22 .
drwxrwsrwx  4 rkuppus  camuser        2048 Nov 27 05:15 ../

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ pwd
/u/rkuppus/ramesh/sachin

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$
```

File Management Utilities

Directory Management

<code>cd</code>	<i>Change working Directory</i>
<code>cd..</code>	Parent Directory
<code>cd.</code>	Current Directory
e.g.,	
<code>\$ cd /usr/trg/c</code>	(current Directory is <i>c</i>)
<code>\$ cd ..</code>	(current Directory is <i>trg</i>)
<code>\$ cd ./c</code>	(current Directory is again <i>c</i>)
or <code>\$ cd c</code>	
<code>\$ cd</code>	(home directory - in this case <i>/usr/trg</i>)

<code>mkdir</code>	<i>Make a Directory</i>
	<code>\$ mkdir pathname</code>

Makes Directory in 777 mode

Write permission should at least be permitted for owner in parent Directory

e.g.,

`$ mkdir /usr/trg2` (makes directory *trg2*)

48

File Management Utilities ...Contd...

`rmdir` *Remove a Directory*

`$ rmdir pathname`

* Directory should be empty, or else

`rm -r` (recursively remove)

e.g.,

`$ rmdir /usr/trg2` (removes directory *trg2*)

`pwd` *Print Working Directory*

File Contents

cat *Concatenate & Print on screen or printer*

\$cat [Options] [Arguments]

Options - take input from stdin

-n no. of output lines

-s squeeze adj. blank lines

-v enable display of non-printing characters

-b used with -n to avoid numbering blank lines

e.g.,

\$ cat try.c Display the contents of **try.c** **on** the screen

\$ cat Takes input from stdin i.e. keyboard and displays **on screen**

\$ cat f1 > f2 Takes input from file **f1** & puts it on
file **f2**

\$ cat f2 > f3 **f3** contains the contents of **f1** **\$ cat f4 >> f3**
Appends the contents of **f4** to
file **f3**

\$ cat try[0-3] > final The file **final** contains contents
of **try0, try1, try2 try3**

\$ cat test* > report The file **report** contains all files
beginning with **test**

ls[Options] *List the Directory Contents*

Options -1 number one single column output

-l long format (**ll** also used)

-a all entries including dot files

-s gives no. of disk blocks

- i** inode no.
- t** ordered by modification time recent first
- R** recursively display all directories, starting from specified or current directory

Examples

\$ ls -I List the files along with the protection bits and the user

\$ ls -a List the files starting with .and..also

\$ ls -l

```
syntab.c
syntab.o
treegen
test
```

\$ ls -I

```
-rw-r--r-- 1 smj proj1 30766 Mar 3 15:08 syntab.c
-rw-r--r-- 1 smj proj1 8759 Mar 3 15:12 syntab.o
-rwxr-xr-x 4 smj proj1 40743 Mar 3 15:23 treegen
drwxrwxr-x 1 smj proj1 53 Mar 1 09:15 test
```

\$ ls -a

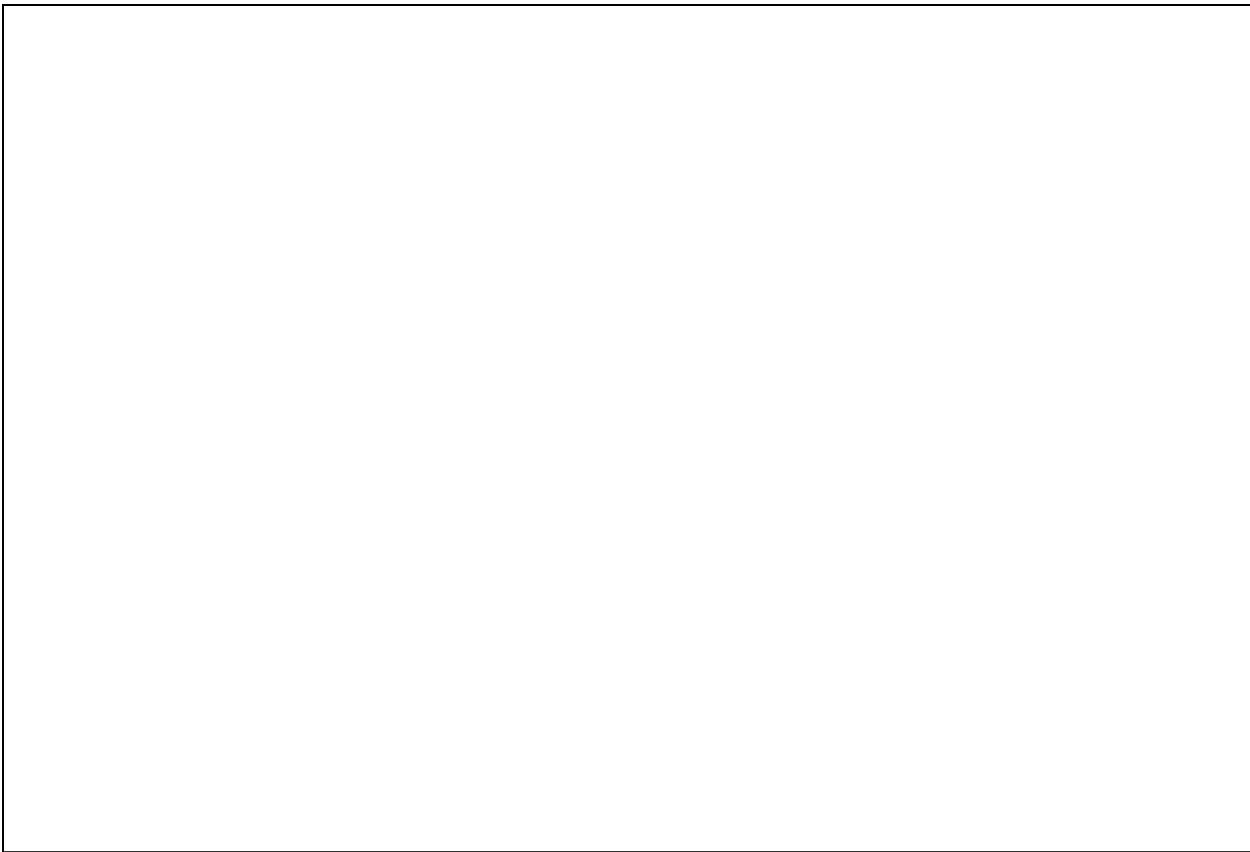
```
. 
..
.profile
.cshrc
syntab.c
```

...

\$ ls -i

```
10936-rw-r--r--I smj proj1 3076 Mar 3 15:08 test.c
```

10936 - inode number of file test.c



```
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cat>cricket
S.no employname    Experiace
1     Arjun          2
2     Ajay           1
3     Kumar          4
4     Suresh         2

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 24
drwxrwsrwx  2 rkuppus  camuser        512 Nov 27 05:34 ../
drwxrwsrwx  4 rkuppus  camuser      2048 Nov 27 05:15 ../
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:38 cricket

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cat cricket
S.no employname    Experiace
1     Arjun          2
2     Ajay           1
3     Kumar          4
4     Suresh         2

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ wc cricket
      5      15     123 cricket
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$
```

lines words characters filename

file

Determine file types

\$file [Options] [Arguments]

Options

-f filelist

Normal File Types

- C program text
- assembler program text
- commands text
- ASCII text
- English text

e.g., **\$ file test.c**

C Program test

cp *copy a file*

-i - user interactive mode

e.g.,

\$ cp test.c test.c.bak

test.c and **test.c.bak** contain the same contents

Extra disk storage

In *Create link*

e.g.,

\$ ln first.c second.c

The file is referenced by two different names

No Extra disk storage

mv Moves or renames files and directories

-i interactive mode

e.g.,

\$ mv old.c new.c

Renames the file **old.c** as **new.c**

rm removes files and directories

-i remove interactively

-f forcible remove

-r remove recursively

- Dangerous

- used in conjunction with **-i**

touch *Updates access, modification or change times of a file*

-a update access time

-m update modification time

-c prevents creating the file

e.g.,

\$ touch f1

- * The current system date & time stamp is put on the file **f1**
- * If **f1** does not exist then it is created with **0 bytes**

The screenshot shows a PuTTY terminal window titled "169.172.51.17 - PuTTY". The session is running on a Linux system with the command-line interface. The user has performed the following sequence of commands:

```
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 24
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 05:55 .
drwxrwsrwx  4 rkuppus  camuser     2048 Nov 27 05:15 ..
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:38 cricket
cp          mv ----changing the name
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cp cricket newcricket

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 32
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 05:55 .
drwxrwsrwx  4 rkuppus  camuser     2048 Nov 27 05:15 ..
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:38 cricket
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:55 newcricket

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ mv cricket oldcricket

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 32
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 05:56 .
drwxrwsrwx  4 rkuppus  camuser     2048 Nov 27 05:15 ..
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:55 newcricket
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:38 oldcricket

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$
```

The terminal window is part of a Windows desktop environment, as evidenced by the taskbar at the bottom which includes icons for Microsoft Word, Microsoft Excel, and Paint.

```

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 32
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 05:56 .
drwxrwsrwx  5 rkuppus  camuser     2048 Nov 27 05:59 ..
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:55 newcricket
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:38 oldcricket

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cd ../lara

[rkuppus@campdb0602: /u/rkuppus/ramesh/lara]
$ ls
total 16
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 05:59 .
drwxrwsrwx  5 rkuppus  camuser     2048 Nov 27 05:59 ..

[rkuppus@campdb0602: /u/rkuppus/ramesh/lara]
$ mv /u/rkuppus/ramesh/sachin/oldcricket lara_cricket

[rkuppus@campdb0602: /u/rkuppus/ramesh/lara]
$ ls
total 24
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 06:01 .
drwxrwsrwx  5 rkuppus  camuser     2048 Nov 27 05:59 ..
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:38 lara_cricket

[rkuppus@campdb0602: /u/rkuppus/ramesh/lara]
$ cd ../sachin

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 24
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 06:01 .
drwxrwsrwx  5 rkuppus  camuser     2048 Nov 27 05:59 ..
-rw-rw-rw-  1 rkuppus  camuser     123 Nov 27 05:55 newcricket

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ 

```

File Comparison

cmp *Compare two files*

If files are same no output is sent to the terminal, **or else** The line number and the byte at which the first difference occurs is reported

-s Outputs nothing Registers return code

Return code

- 0 if files are identical
- 1 if files are different
- 2 on error

e.g.,

```
$ cmp test1 test2  
test1 and test2 differ in char 36 line 3  
$ cmp -s test1 test2  
$ echo $status  
outputs 1 indicating that the files are different
```

diff - *Reports more than one differences*

```
$diff [Options] file1 file2
```

- b Ignores trailing blanks
- e Gives a list of **ed** commands so as to convert file1 into file2.

e.g.,

```
$ diff test1 test2
```

Outputs: n1 a n3,n4
n1,n2 d n3
n1,n1 c n3,n4

where * n1 ,n2, n3 ,n4 are line numbers

* a ,d, c means **append**, **delete** ,**change** respectively

comm *Display common lines*

```
$comm -[123] f1 f2
```

Prints a three column output:

- lines that occur only in f1
- lines that occur only in f2
- lines that occur in both

comm -12 - prints lines common to the two files

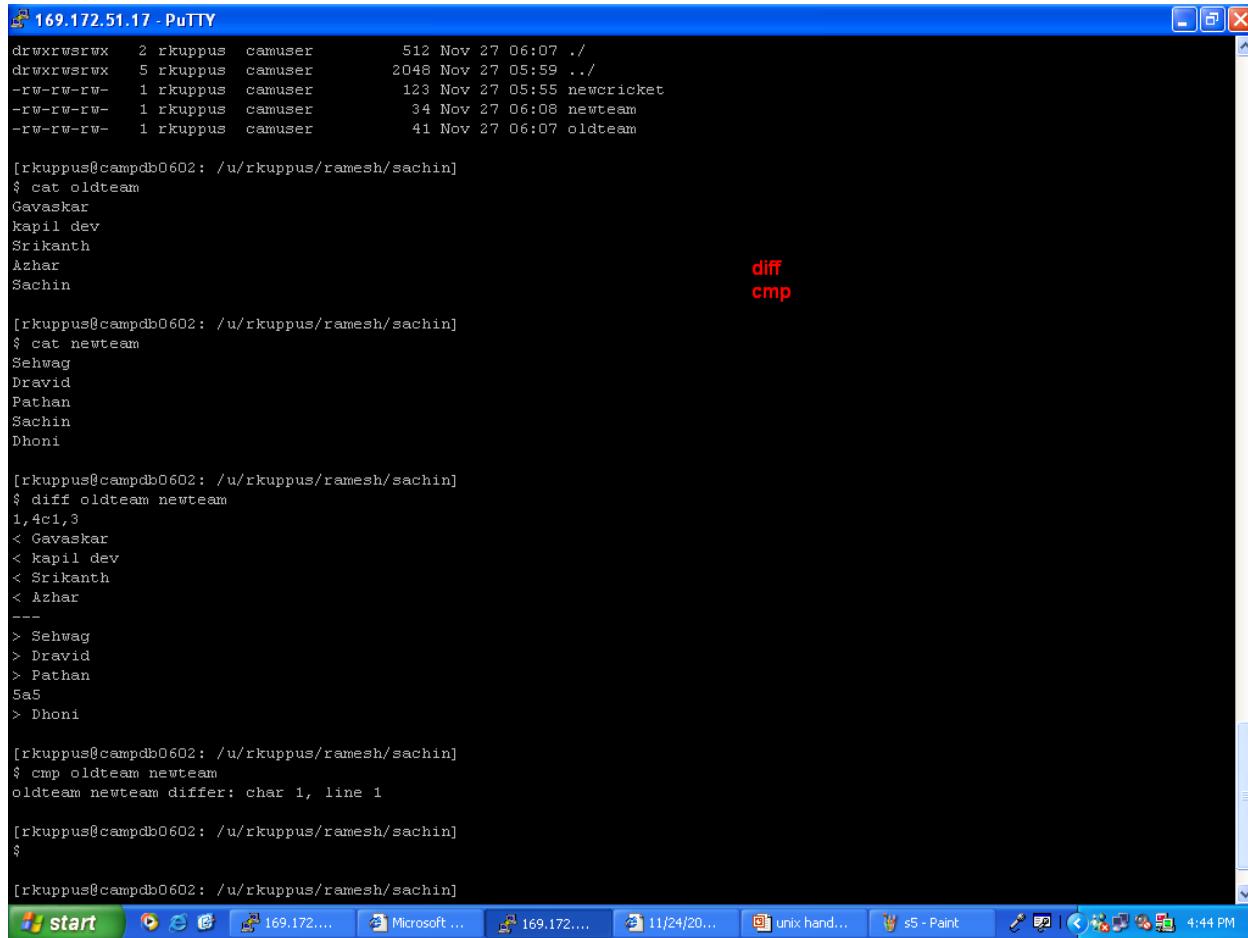
comm -23 - prints only lines in the first file but not in the second

comm -123 - prints nothing

e.g.,

\$ comm test1 test2 Reports the common lines between files
test1, test2 and reports the lines differing

\$ comm -12 test1 test2 Prints line common to both



The screenshot shows a PuTTY terminal window titled '169.172.51.17 - PuTTY'. The session content is as follows:

```
drwxrwsrwx 2 rkuppus camuser 512 Nov 27 06:07 ./
drwxrwsrwx 5 rkuppus camuser 2048 Nov 27 05:59 ../
-rw-rw-rw- 1 rkuppus camuser 123 Nov 27 05:55 newcricket
-rw-rw-rw- 1 rkuppus camuser 34 Nov 27 06:08 newteam
-rw-rw-rw- 1 rkuppus camuser 41 Nov 27 06:07 oldteam

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cat oldteam
Gavaskar
Kapil Dev
Srikanth
Azhar
Sachin

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cat newteam
Sehwag
Dravid
Pathan
Sachin
Dhoni

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ diff oldteam newteam
1,4c1,3
< Gavaskar
< Kapil Dev
< Srikanth
< Azhar
---
> Sehwag
> Dravid
> Pathan
> Sachin
> Dhoni

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cmp oldteam newteam
oldteam newteam differ: char 1, line 1

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$
```

Red text 'diff' and 'cmp' are highlighted in the terminal output.

pack

Compress the file

\$ pack <filename>

e.g., \$ pack try

- Creates a file try.z which is packed
- Normally the executables are packed
- The size is reduced by 25 - 40 %

unpack *Uncompress packed file*

or

pcat

e.g., \$ unpack try.z

or

\$ pcat try.z

unpacks the file try.z

MOUNT

mount *Associates a directory with a device*

e.g., Mounting a floppy on the root file system

umount *Dissociates directory from the device*

e.g.,

\$ mount /dev/fd096 /mnt Mounts the floppy on

the directory **/mnt**

\$ umount /mnt Dissociates **/mnt** from
the floppy

FILE SECURITY

passwd *To change the password*

chown *To change the ownership of the file*

\$ chown owner filename

e.g., **\$ chown trg2 test.c**

- * Initially the owner is trg1
- * Only the owner or the superuser can change the ownership of the file

chmod *change the permissions of the file*

\$ chmod who op permission <filelist>

who **a, u, g, o** all, user, group, others

op **+, -, =** + add, - remove, = set

permission **r,w,x** **r read, w write, x execute**

e.g.,

\$ chmod a=rw test.c

* users, group, others have **read**

and **write** permissions

\$ chmod u+r, g+w, o+x test.c

* **read** for users **write** for groups **execute** for others

\$ chmod 777 test.c

* Sets **read, write, execute** Permissions

umask *Set file creation mode mask*

\$ umask nnn (nnn set file creation mode)

umask can also be set as a *shell variable*

e.g.,

umask 022

- Files normally created with 777 mode is assigned
755 permission

The value of each digit is subtracted from the corresponding "digit" specified by the system for the creation of a file.

tail *Displays the last lines of file*

options : -n (n= no. of lines)

e.g., **\$ tail -30 test.c**

Displays the last 30 lines of file **test. c**

head *Displays the top lines of file*

e.g., **\$ head -10 test.c** Displays the first 10 lines of **test.c**

split *Splits the file into different files as specified by the number of lines*

e.g., **\$ split -20 test.c**

Splits the file **test.c** in blocks of 20 lines and creates files **xaa, xab, xac**

and so on, such that

xaa has first 20 lines of **test.c**

xab has the next 20 lines of **test.c**

...

The file test.c is unaffected

\$ split-20 test.c try Generates files as **tryaa , tryab , tryac**

paste *Joins the two or more files horizontally*

e.g., **\$ paste xaa xab**

File **xaa** and **xab** are joined horizontally and output to the terminal

169.172.51.17 - PuTTY

```
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 32
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 06:20 .
drwxrwsrwx  5 rkuppus  camuser     2048 Nov 27 05:59 ..
-rw-rw-rw-  1 rkuppus  camuser       66 Nov 27 07:12 newteam
-rw-rw-rw-  1 rkuppus  camuser      41 Nov 27 06:20 oldteam

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cat newteam
Sehwag
Tendulkar
Dravid
dhoni
kaif
Yuvraj
pathan
rp.singh
H.singh

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ head -4 newteam
Sehwag
Tendulkar
Dravid
dhoni

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ tail -3 newteam
pathan
rp.singh
H.singh

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$
```



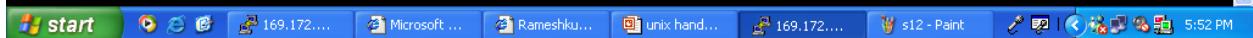
169.172.51.17 - PuTTY

```
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls -lrt
total 32
drwxrwsrwx  5 rkuppus  camuser      2048 Nov 27 05:59 ../
-rw-rw-rw-  1 rkuppus  camuser       41 Nov 27 06:20 oldteam
-rw-rw-rw-  1 rkuppus  camuser      66 Nov 27 07:12 newteam
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:14 football
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:15 basketball
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:15 boxing
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:15 tabletennis
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 07:15 ./

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls -lrt|head -5
total 32
drwxrwsrwx  5 rkuppus  camuser      2048 Nov 27 05:59 ../
-rw-rw-rw-  1 rkuppus  camuser       41 Nov 27 06:20 oldteam
-rw-rw-rw-  1 rkuppus  camuser      66 Nov 27 07:12 newteam
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:14 football

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls -lrt|tail -6
-rw-rw-rw-  1 rkuppus  camuser      66 Nov 27 07:12 newteam
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:14 football
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:15 basketball
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:15 boxing
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:15 tabletennis
drwxrwsrwx  2 rkuppus  camuser      512 Nov 27 07:15 ./

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ 
```



```
[ckuppus@campd0402: /u/ckuppus/c næsh/machin]
$ ls -lrt|head -5
total 32
drwxrwxrwx  5 ckuppus  ckuppus  2048 Nov 27 05:59 ../
-rw-rw-rw-  1 ckuppus  ckuppus   41 Nov 27 06:20 oldteam
-rw-rw-rw-  1 ckuppus  ckuppus   66 Nov 27 07:12 newteam
-rw-rw-rw-  1 ckuppus  ckuppus    0 Nov 27 07:14 football
drwxrwxrwx  2 ckuppus  ckuppus  512 Nov 27 07:15 ./

[jkuppus@campd0402: /u/ckuppus/c næsh/machin]
$ ls -lrt|tail -4
-rw-rw-rw-  1 ckuppus  ckuppus   66 Nov 27 07:12 newteam
-rw-rw-rw-  1 ckuppus  ckuppus    0 Nov 27 07:14 football
-rw-rw-rw-  1 ckuppus  ckuppus   90 Nov 27 07:15 basketball
-rw-rw-rw-  1 ckuppus  ckuppus   90 Nov 27 07:15 boxing
-rw-rw-rw-  1 ckuppus  ckuppus   90 Nov 27 07:15 tabletennis
drwxrwxrwx  2 ckuppus  ckuppus  512 Nov 27 07:15 ./

[jkuppus@campd0402: /u/ckuppus/c næsh/machin]
$ ls -lrt|head -6 >list1
[jkuppus@campd0402: /u/ckuppus/c næsh/machin]
$ cat list1
total 32
drwxrwxrwx  5 ckuppus  ckuppus  2048 Nov 27 05:59 ../
-rw-rw-rw-  1 ckuppus  ckuppus   41 Nov 27 06:20 oldteam
-rw-rw-rw-  1 ckuppus  ckuppus   66 Nov 27 07:12 newteam
-rw-rw-rw-  1 ckuppus  ckuppus    0 Nov 27 07:14 football
-rw-rw-rw-  1 ckuppus  ckuppus   90 Nov 27 07:15 basketball
drwxrwxrwx  2 ckuppus  ckuppus  512 Nov 27 07:15 ./

[jkuppus@campd0402: /u/ckuppus/c næsh/machin]
$ ls -lrt|tail -6 > list2
[jkuppus@campd0402: /u/ckuppus/c næsh/machin]
$ cat list2
total 32
-rw-rw-rw-  1 ckuppus  ckuppus    0 Nov 27 07:15 basketball
-rw-rw-rw-  1 ckuppus  ckuppus    0 Nov 27 07:15 boxing
-rw-rw-rw-  1 ckuppus  ckuppus    0 Nov 27 07:15 tabletennis
-rw-rw-rw-  1 ckuppus  ckuppus  334 Nov 27 07:19 list1
-rw-rw-rw-  1 ckuppus  ckuppus    0 Nov 27 07:19 list2
drwxrwxrwx  2 ckuppus  ckuppus  512 Nov 27 07:19 ./

[jkuppus@campd0402: /u/ckuppus/c næsh/machin]
```

169.172.51.17 - PuTTY

```
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cat newteam
Sehwag
Tendulkar
Dravid
dhoni
kaif
Yuvraj
pathan
rp.singh
H.singh

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ split -4 newteam

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 72
drwxrwsrwx  2 rkuppus  camuser      512 Nov 28 00:34 ./
drwxrwsrwx  5 rkuppus  camuser    2048 Nov 27 05:59 ../
-rw-rw-rw-  1 rkuppus  camuser       0 Nov 27 07:15 basketball
-rw-rw-rw-  1 rkuppus  camuser       0 Nov 27 07:15 boxing
-rw-rw-rw-  1 rkuppus  camuser       0 Nov 27 07:14 football
-rw-rw-rw-  1 rkuppus  camuser     387 Nov 27 07:19 liat2
-rw-rw-rw-  1 rkuppus  camuser     334 Nov 27 07:19 list1
-rw-rw-rw-  1 rkuppus  camuser      66 Nov 27 07:12 newteam
-rw-rw-rw-  1 rkuppus  camuser     41 Nov 27 06:20 oldteam
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:15 tabletennis
-rw-rw-rw-  1 rkuppus  camuser     30 Nov 28 00:34 xaa
-rw-rw-rw-  1 rkuppus  camuser     28 Nov 28 00:34 xab
-rw-rw-rw-  1 rkuppus  camuser      8 Nov 28 00:34 xac

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cat xaa
Sehwag
Tendulkar
Dravid
dhoni

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$
```

169.172.51.17 - PuTTY

```
[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cat newteam
Sehwag
Tendulkar
Dravid
dhoni
kaif
Yuvraj
pathan
rp.singh
H.singh

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ split -4 newteam

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ ls
total 72
drwxrwsrwx  2 rkuppus  camuser      512 Nov 28 00:34 .
drwxrwsrwx  5 rkuppus  camuser    2048 Nov 27 05:59 ..
-rw-rw-rw-  1 rkuppus  camuser       0 Nov 27 07:15 basketball
-rw-rw-rw-  1 rkuppus  camuser       0 Nov 27 07:15 boxing
-rw-rw-rw-  1 rkuppus  camuser       0 Nov 27 07:14 football
-rw-rw-rw-  1 rkuppus  camuser     387 Nov 27 07:19 liat2
-rw-rw-rw-  1 rkuppus  camuser     334 Nov 27 07:19 list1
-rw-rw-rw-  1 rkuppus  camuser      66 Nov 27 07:12 newteam
-rw-rw-rw-  1 rkuppus  camuser      41 Nov 27 06:20 oldteam
-rw-rw-rw-  1 rkuppus  camuser      0 Nov 27 07:15 tabletennis
-rw-rw-rw-  1 rkuppus  camuser     30 Nov 28 00:34 xaa
-rw-rw-rw-  1 rkuppus  camuser     28 Nov 28 00:34 xab
-rw-rw-rw-  1 rkuppus  camuser      8 Nov 28 00:34 xac

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$ cat xaa
Sehwag
Tendulkar
Dravid
dhoni

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]
$
```

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]

\$ who

rkuppus	pts/0	Nov 28 00:21	(157.227.91.168)
rkuppus	pts/3	Nov 27 22:56	(157.227.90.95)
rkuppus	pts/6	Nov 28 00:35	(157.227.91.168)
rkuppus	pts/5	Nov 28 00:31	(157.227.90.136)

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]

\$ finger

Login	Name	TTY	Idle	When	Site Info
rkuppus	KUPPUSWAMY	*p0	1	Mon 00:21	RAMESHKUMAR,1000245694
rkuppus	KUPPUSWAMY	*p3	3	Sun 22:56	RAMESHKUMAR,1000245694
rkuppus	KUPPUSWAMY	*p6		Mon 00:35	RAMESHKUMAR,1000245694
rkuppus	KUPPUSWAMY	*p5		Mon 00:31	RAMESHKUMAR,1000245694

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]

\$ whoami

rkuppus

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]

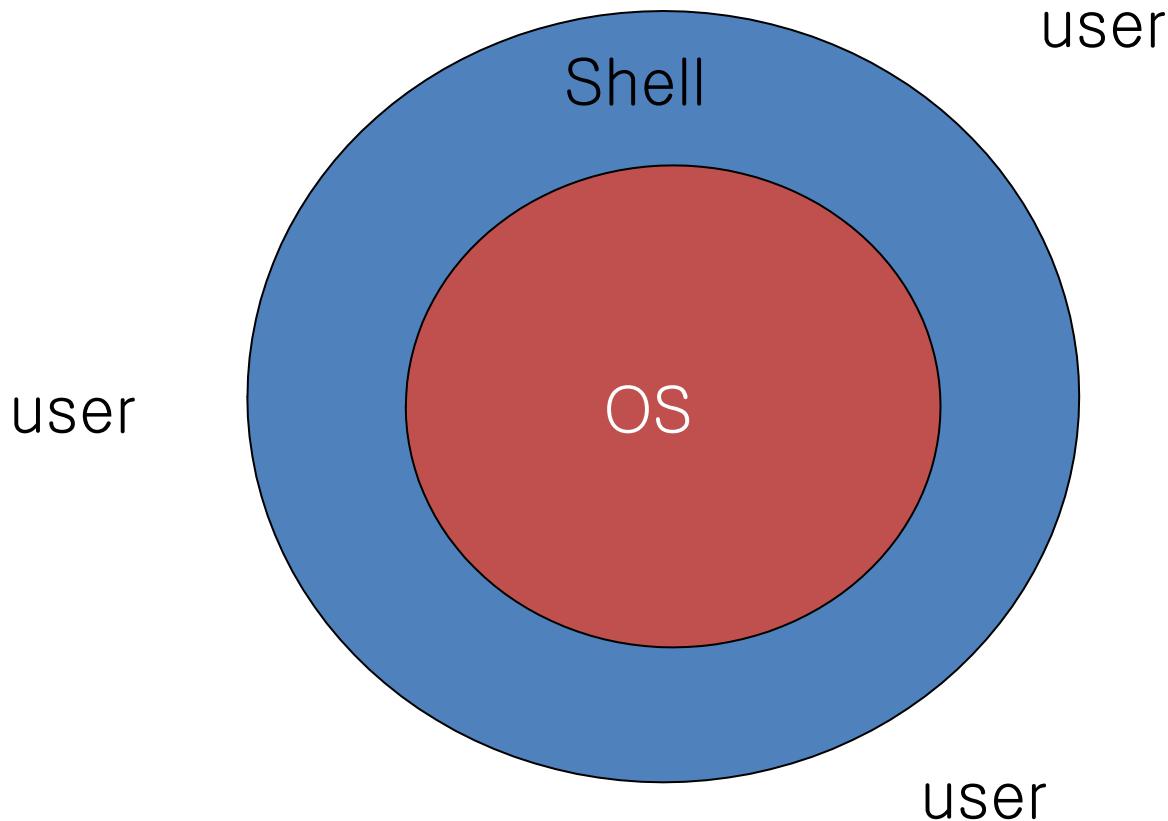
\$ cal

November 2005						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
	1	2	3	4	5	
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

[rkuppus@campdb0602: /u/rkuppus/ramesh/sachin]

\$

SHELL



SHELL SCRIPT

- A command programming language that executes commands read from a terminal or a file
- Uses built-in commands/syntax and OS programs
- Why use shell scripts?
 - Can do repetitive tasks easily & quickly

CREATE A SHELL SCRIPT

- ❖ Creating a simple shell script
 - A shell script is a file that contains commands that the shell can execute.
 - Any commands you enter in response to a shell prompt.

- A utility
- A compiled program
- Another shell script
- Control flow commands
- ❖ Run a shell script
 - Enter the script filename on the command line
 - The shell interprets and execute the commands one after another
- ❖ Why shell script?
 - Simply and quickly initiate a complex series of tasks or a repetitive procedure.

SHELL SCRIPT PROGRAMMING

- ❖ Make the file executable
 - When you create a shell script using a editor, does it have execute permission typically?
 - Example: (Make sure you are using tcsh/csh script!...)

```

willow> echo $SHELL
/bin/tcsh
willow> ./test
./test: Permission denied.

willow> ls -l test
-rw-r--r-- 1 student ums      33 Sep 18 16:33 test
willow> chmod +x test
willow> ./test
This is Test!

```

INVOKE THE SHELL SCRIPT

- ❖ Give the shell a command on the command line
 - The shell forks a process
 - Which creates a duplicate of the shell process (subshell)
 - The new process attempt to exec the command
 - If the command is an executable program
 - Exec succeeds
 - System overlays the newly created subshell with the executables program
 - The command is a shell script
 - Exec failed
 - The command is assumed to be a shell script
 - The subshell runs the commands in the shell.
- Shell itself is program,
 - It can be run as a command in a shell and also accepts arguments.
Note: Let's find your default shell executing "echo \$SHELL"

willow> echo \$SHELL

/bin/tcsh

- To run a shell script
 - Which does not have executable permission

Ex: willow>tcsh test

- Run the script with different shell other than your interactive shell

Ex: willow>sh test

- Put special characters on the first line of a shell script
 - To tell OS checks what kind of file it is before attempting to exec it
 - To tell which utility to use (sh, csh, tcsh, ...)

- Special sequence
 - The first two characters of a script are #!
 - Then followed by the absolute pathname of the program that should execute the script

Ex:

```
willow> more test  
#!/bin/tcsh  
# This line will not run since it is commented out...  
echo 'This is Test!'
```

COMMENTS

- Comments make shell scripts easier to read and maintain
- Pound sign (#) starts a comment line until the end of that line as second line in previous example, except
 - #! In the first line.
 - Or inside quotes

BASIC SYNTAX

(list)

`list`

{ list; }

name=value

name () { list; }

\$name

\${name}

```
#!/bin/sh

myvar="myvalue"
if [ "$myvar" = "" ]; then
    echo "nothing!";
else
    echo "you've got $myvar"
fi
```

FLOW CONTROL FOR

- for name [in word ...]
do list done
- #!/bin/sh

```
for i in 5 4 3 2 1;
do echo "countdown: $i";
done
```

FLOW CONTROL WHILE

- while list;
do list;
done
- #!/bin/sh

```
unit="x"

while [ "$string" != "xxxxxx" ]; do
string=$string$unit;
echo "not yet~ ($string)";
done
```

```
echo "ok~ got $string"
```

FLOW CONTROL CASE

- case word in
[pattern [| pattern]) list ;;] ... esac

```
#!/bin/sh
```

```
case $1 in
    hi) echo "hello~";;
    bye) echo "byebye~";;
esac
```

PREDEFINED VARIABLES

- # : number of arguments
- ? : last command's return value
- \$: process id of this shell
- ! : process id of last background command
- 1, 2, 3, ... 9 : n-th argument

EXAMPLE

```
#!/bin/sh

# shortcut script

case $1 in
    a) telnet ara;;
    s) telnet ska;;
    l) telnet loco;;
    n) telnet noah;;
    m) mutt

    t) if [ -f $HOME/TODO ]; then
        cat $HOME/TODO | more;
    fi;;
esac
```

esac

PARAMETERS AND VARIABLES

- A shell parameter is associated with a value that is accessible to the user.
 - Shell variables
 - Names consist of letters, digits and underscore
 - By convention, environment variables uses uppercase
 - User created variables (create and assign value)
 - Keyword shell variables
 - Has special meaning to the shell
 - Being created and initialized by the startup file
 - Positional parameters
 - Allow you to access command line arguments
 - Special parameters
 - Such as
 - The name of last command
 - The status of most recently executed command
 - The number of command-line arguments

POSITIONAL PARAMETERS

- The command name and arguments are the positional parameters.
 - Because you can reference them by their position on the command line
 - \$0 : Name of the calling program

- \$1 - \$9 : Command-line Arguments
 - The first argument is represented by \$1
 - The second argument is represented by \$2
 - And so on up to \$9
 - The rest of arguments has to be shifted to be able to use \$1- \$9 parameters.

EXAMPLE

- Change directory to your assigned numbered subdirectory

willow> cd 1

- List the directory contents, confirming display_5args

willow> ls -l display_5args

- Change mode of display_5args to executable

willow> chmod +x display_5args

- Execute the script

willow> ./display_5args 1 2 3 4 5

you are running script ./display_5args with parameter 1 2 3 4 5

- \$1-\$9 allows you to access 10 arguments
 - How to access others?
- Promote command-line arguments: shift
 - Built-in command shift promotes each of the command-line arguments.
 - The first argument (which was \$1) is discarded
 - The second argument (which was \$2) becomes \$1
 - The third becomes the second
 - And so on
 - Makes additional arguments available

- Repeatedly using shift is a convenient way to loop over all the command-line arguments

➤ Example:

```
willow> more demo_shift
```

```
#!/bin/tcsh
```

```
echo $1 $2 $3
```

```
shift
```

```
echo $1 $2
```

```
shift
```

```
echo $1
```

```
willow> ./demo_shift 1 2 3
```

```
1 2 3
```

```
2 3
```

```
3
```

```
willow> more demo_shift
```

```
#!/bin/tcsh
```

```
echo $1 $2 $3
```

```
shift
```

```
echo $1 $2
```

```
shift
```

```
echo $1
```

```
shift
```

```
echo $?
```

```
shift
```

```
echo $?
```

```
shift
```

```
echo $?  
willow> ./demo_shift 1 2 3 4  
1 2 3  
2 3  
3  
0  
0  
shift: No more words
```

SPECIAL PARAMETERS

- Useful values
 - Command-line arguments
 - Execution of shell commands
 - Can not change the value directly, like positional parameters
- Value of Command-line arguments: \$* and \$@
 - \$* and \$@ represent all the command_line arguments (not just the first nine)
 - "\$*" : treat the entire list of arguments as a single argument
 - "\$@" : produce a list of separate arguments (Only bash/ksh/sh)

BASH SCRIPT WITH \$*and \$@

```
willow> more for_test.bash  
#!/bin/bash  
echo "using \$* "
```

```
for arg in "$*"
do
    echo "$arg"
done

echo "using \$@"
for arg in "$@"
do
    echo "$arg"
done
```

willow> ./for_test.bash 1 2 3

using \$*

1 2 3

using \$@

1

2

3

TCSH SCRIPT WITH \$* and \$@

```
willow> more for_test
#!/bin/tcsh
echo 'using $*'
```

```
foreach arg ($*)
    echo "$arg"
end
echo 'using $@'
foreach arg ($@)
    echo "$arg"
end
willow> ./for_test 1 2 3
using $*
1
2
3
using $@
Illegal variable name.
```

SPECIAL PARAMETERS

- The number of arguments: \$#
 - Return a decimal number
 - Use the test to perform logical test on this number

```
willow> more num_args
echo this script is called with $# arguments.
willow> chmod +x num_args
willow> ./num_args
```

this script is called with 0 arguments.

```
willow> ./num_args 1
```

this script is called with 1 arguments.

```
willow> ./num_args 2
```

this script is called with 1 arguments.

```
willow> ./num_args 0
```

this script is called with 1 arguments.

➤ **Exit status: \$?**

- When a process stops executing for any reason, it returns an exit status to its parent process.
- By convention,
 - Nonzero represents a false value that the command failed.
 - A zero value is true and means that the command was successful
- You can specify the exit status that a shell script returns by using the `exit` built-in followed by a number
 - Otherwise, the exit status of the script is the exit status of the last command the script ran.

- `willow> ls a`
- `a: No such file or directory`
- `willow> echo $?`
- `2`
- `willow> echo olemiss`
- `olemiss`
- `willow> echo $?`

- 0
- willow> more exit_status
- echo this program will have the exit code of 8.
- exit 8
- willow> ./exit_status
- this program will have the exit code of 8.
- willow> echo \$?
- 8
- willow> echo \$?
- 0

SUMMARY

- A shell is both a command interpreter and a programming language.
- Job control
 - Control-z/fg/bg/&
- Variables
 - Local and environment variables
 - Declare and initialize a variable (no type)
 - Export unset
- Command line expansion
 - Parameter expansion/variable expansion/command/substitution/pathname expansion
 - Quote (`` " " \)
 - " " all but parameter, variable expansion and \
 - ' ' suppress all types of expansion
 - \ escaping the following special character

BASIC SCRIPT EXAMPLE

```
willow> more basic_script

#!/bin/tcsh

echo 'Listing the files in long format appending due date/time'

echo

ls -lrtah

echo

echo 'Listing the files in long format appending due date/time'

echo

df -k

# Using diff to find two files differences and writing them to another file

diff -c for_test.bash for_test >> file_differences &

echo

echo 'sleeping mode for 4 seconds. Please wait!'

echo

sleep 4

echo

echo 'GO REBELS'

echo 'To find out the differences of files for_test and for_test.bash, '

echo 'Please open file_differences via using cat command as shown below:'

echo 'cat file_differences'
```

PIPELINING

- Syntax : command1 | command2
- Connecting one process(command1)'s output to another process(command2)'s input

- Can do complex jobs by combining many small programs
- Examples
 - `ls -l | sort`
 - `finger | cut -d " " -f 3,4,5 | wc`

REDIRECTION

- Syntax : command <|<<|>|>> path
- Connecting process's output/input to a given file
- Can save any output to a file and can load any file to a program input
- Examples
 - `du -sh * > du_dump.file`
 - `find ~ -name .*rc > rcfies`
 - `sort < unsorteddata > sorteddata`
 - `date >> mydatelogfile`
 -

JOB CONTROL

- Job control is a way to do multitasking with the command-line interface
- Syntax and commands
 - `command &`
 - `bg [%n]`
 - `fg [%n]`
 - `jobs`
 - `kill [%n]`

SAMPLE SHELL SCRIPT

- vi myfirstscript.sh
- ```
#! /bin/csh

set directory= `pwd`
echo The date today is `date`
echo The current directory is $directory
```
- chmod u+x myfirstscript.sh
  - myfirstscript.sh

## **USING SHELL SCRIPT**

- UNIX shell scripts are text files that contain sequences of UNIX commands
- Like high-level source files, a programmer creates shell scripts with a text editor
- Shell scripts do not have to be converted into machine language by a compiler
- This is because the UNIX shell acts as an interpreter when reading script files
- Further, the chmod command tells the computer who is allowed to use the file: the owner (u), the group (g), or all other users (o)
- Shell programs run less quickly than do compiled programs, because the shell must interpret each UNIX command inside the executable script file before it is executed

## **programming features of the UNIX shell:**

- *Shell variables*
- *Operators*

- *Logic structures*

## **programming features of the UNIX shell:**

- *Shell variables*: Your scripts often need to keep values in memory for later use. Shell variables are symbolic names that can access values stored in memory
- *Operators*: Shell scripts support many operators, including those for performing mathematical operations
- *Logic structures*: Shell scripts support sequential logic (for performing a series of commands), decision logic (for branching from one point in a script to another), looping logic (for repeating a command several times), and case logic (for choosing an action from several possible alternatives)

## VARIABLES

- Variables are symbolic names that represent values stored in memory
- The three types of variables discussed in this section are configuration variables, environment variables, and shell variables
- Use configuration variables to store information about the setup of the operating system, and do not change them
- You can set up environment variables with initial values that you can change as needed
- These variables, which UNIX reads when you log in, determine many characteristics of your session
- Shell variables are those you create at the command line or in a shell script
- Environment and configuration variables bear standard names, such as HOME, PATH, SHELL, USERNAME, and PWD

(Configuration and environment variables are capitalized to distinguish them from user variables)

- vi myinputs.sh
- ```
#! /bin/csh
echo Total number of inputs: $#argv
```

```
echo First input: $argv[1]
echo Second input: $argv[2]
```

- chmod u+x myinputs.sh
- myinputs.sh HUSKER UNL CSE

To see a list of your environment variables:

```
$ printenv
```

or:

```
$ printenv | more
```

SHELL OPERATORS

- The Bash shell operators are divided into three groups: defining and evaluating operators, arithmetic operators, and redirecting and piping operators
- expr supports the following operators:
 - arithmetic operators: +,-,*,/,%
 - comparison operators: <, <=, ==, !=, >=, >
 - boolean/logical operators: &, |
 - parentheses: (,)
 - precedence is the same as C, Java

EXAMPLE

- vi math.sh

```
#!/bin/csh
```

```
set count=5  
      set count=`expr $count + 1`  
      echo $count  
chmod u+x math.sh  
math.sh
```

LOGIC STRUCTURES

The four basic logic structures needed for program development are:

- Sequential logic
- Decision logic
- Looping logic
- Case logic

Sequential logic states that commands will be executed in the order in which they appear in the program

The only break in this sequence comes when a branch instruction changes the flow of execution

Decision logic enables your program to execute a statement or series of statements only if a certain condition exists

The if statement is the primary decision-making control structure in this type of logic

if-then

```
if ( expr ) simple-command
```

if-then-else

```
if ( expr ) then  
    command-set-1
```

```
[else  
    command-set-2]  
endif
```

- **A simple example**

```
#!/bin/csh  
  
if ($#argv != 2) then  
    echo $0 needs two parameters!  
    echo You are inputting $#argv parameters.  
  
else  
    set par1 = $argv[1]  
    set par2 = $argv[2]  
  
endif
```

Another example:

```
#! /bin/csh  
  
# number is positive, zero or negative  
echo "enter a number:"  
  
set number = $<  
  
if ( $number < 0 ) then  
    echo "negative"  
else if ( $number == 0 ) then  
    echo zero  
else  
    echo positive  
endif
```

```

#!/bin/csh

if {(`grep UNIX $argv[1] > /dev/null`)} then
    echo UNIX occurs in $argv[1]
else
    echo No!
    echo UNIX does not occur in $argv[1]
endif

```

Redirect intermediate results into `>/dev/null`, instead of showing on the screen.

LOOPING LOGIC

- In looping logic, a control structure (or loop) repeats until some condition exists or some action occurs
- You will learn two looping mechanisms in this section: the for loop and the while loop
- You use the for command for looping through a range of values.
- It causes a variable to take on each value in a specified set, one at a time, and perform some action while the variable contains each individual value

WHILE LOOP

- A different pattern for looping is created using the while statement
- The while statement best illustrates how to set up a loop to test repeatedly for a matching condition
- The while loop tests an expression in a manner similar to the if statement
- As long as the statement inside the brackets is true, the statements inside the do and done statements repeat

```
while ( expr )
    command_set
end

foreach var ( wordlist )
    command_set
end
```

LOOPING LOGIC

Program:

```
#!/bin/csh

foreach person (Bob Susan Joe Gerry)
    echo Hello $person
end
```

Output:

Hello Bob

Hello Susan

Hello Joe

Hello Gerry

- **Adding integers from 1 to 10**

```
#!/bin/csh

set i=1
set sum=0
```

```
while ($i <= 10)
    echo Adding $i into the sum.
    set sum=`expr $sum + $i`
    set i=`expr $i + 1`
end
echo The sum is $sum.
```

SWITCH LOGIC

The switch logic structure simplifies the selection of a match when you have a list of choices

It allows your program to perform one of many actions, depending upon the value of a variable

```
switch ( var )
    case string1:
        command_set_1
        breaksw
    case string2:
        command_set_2
        breaksw
    default
        command_set_3
endsw
```

```
#!/bin/csh
if ($#argv == 0 ) then
    echo "No arguments supplied...exiting"
```

```
else
switch ($argv[1])
case [yY]:
    echo Argument one is yes.
    breaksw
case [nN]:
    echo Argument one is no.
    breaksw
default:
    echo Argument one is neither yes nor no.
    breaksw
endsw
endif
```

CONTINUE WITH LINES

```
% echo This \
Is \
A \
Very \
Long \
Command Line
This Is A Very Long Command Line
%
```

EXIT STATUS

- \$?
- 0 is True

```
% ls /does/not/exist
```

```
% echo $?
```

```
1
```

```
% echo $?
```

```
0
```

EXIT STATUS EXIT

```
% cat > test.sh <<_TEST_
```

```
exit 3
```

```
_TEST_
```

```
% chmod +x test.sh
```

```
% ./test.sh
```

```
% echo $?
```

```
3
```

LOGIC TEST

```
% test 1 -lt 10
```

```
% echo $?
```

```
0
```

```
% test 1 == 10
```

```
% echo $?
```

```
1
```

```
test
```

```
[ ]
```

```
- [ 1 -lt 10 ]  
[[ ]]  
- [[ "this string" =~ "this" ]]  
(( ))  
- (( 1 < 10 ))
```

```
[ -f /etc/passwd ]  
[ ! -f /etc/passwd ]  
[ -f /etc/passwd -a -f /etc/shadow ]  
[ -f /etc/passwd -o -f /etc/shadow ]
```

An aside: \$(()) for Math

```
% echo $(( 1 + 2 ))
```

```
3
```

```
% echo $(( 2 * 3 ))
```

```
6
```

```
% echo $(( 1 / 3 ))
```

```
0
```

LOGIC : IF

if *something*

then

```
:  
# "elif" a contraction of "else if":  
elif something-else  
then  
:  
else  
then  
:  
fi
```

```
if [ $USER -eq "borwicjh" ]  
then  
:  
# "elif" a contraction of "else if":  
elif ls /etc/oratab  
then  
:  
else  
then  
:  
fi
```

```
# see if a file exists  
if [ -e /etc/passwd ]  
then
```

```
echo "/etc/passwd exists"  
else  
    echo "/etc/passwd not found!"  
fi
```

LOGIC : FOR

```
for i in 1 2 3
```

```
do
```

```
    echo $i
```

```
done
```

```
for i in /*
```

```
do
```

```
    echo "Listing $i:"
```

```
    ls -l $i
```

```
    read
```

```
done
```

```
for i in /*
```

```
do
```

```
    echo "Listing $i:"
```

```
ls -l $i
```

```
read
```

```
done
```

LOGIC C-STYLE FOR

```
for (( expr1 ;
```

```
    expr2 ;
```

```
    expr3 ))
```

```
do
```

```
list
```

```
done
```

```
LIMIT=10
```

```
for (( a=1 ;
```

```
    a<=LIMIT ;
```

```
    a++ ))
```

```
do
```

```
echo -n "$a "
```

```
done
```

LOGIC : WHILE

```
while something
```

```
do
```

:

done

```
a=0; LIMIT=10  
while [ "$a" -lt "$LIMIT" ]  
do  
    echo -n "$a "  
    a=$(( a + 1 ))  
done
```

COUNTERS

```
COUNTER=0  
while [ -e "$FILE.COUNTER" ]  
do  
    COUNTER=$(( COUNTER + 1 ))  
done
```

- Note: race condition

REUSING CODE : “ SOURCING”

```
% cat > /path/to/my/passwords <<_PW_  
FTP_USER="sct"
```

```
_PW_
% echo $FTP_USER
% . /path/to/my/passwords
% echo $FTP_USER
sct
%
```

VARIABLE MANIPULATION

```
% FILEPATH=/path/to/my/output.lis
% echo $FILEPATH
/path/to/my/output.lis
% echo ${FILEPATH%.lis}
/path/to/my/output
% echo ${FILEPATH#*/}
path/to/my/output.lis
% echo ${FILEPATH##*/}
output.lis
```

