T A N G R A M

3-way TANGRAM
User Manual

Version 8.2 for Personal Computers
running Windows (any version)

Ref. TG14E - Revision 1.9

Contact

CODEWORK Italia srl, Corso Lanza 105, 10133 Torino, Italy
Tel. +39 011 6601560
E-mail: `info@codework.it` `sales@codework.it` `support@codework.it`
Homepage: `www.codework.it/tangram/`

Credits

Cover Design: *Chroma*
Text: *Mauro Guazzo, Marco Bruzzone*
Composition: *Federica Pusineri, Cristina Prono, Valerio Tamburini*

Contents

Chapter 1

Introduction

"Entia non sunt multiplicanda praeter necessitatem"
William of Occam (1285-1349)

This document represents the user manual of the software product **3-way TANGRAM**, a tool for managing and manipulating multidimensional data structures.

It is intended both for the end user and for the application developer.

TANGRAM databases can be accessed from Microsoft Excel via an add-on called **TANGRAM-reader** which is explained in a different document `readerman.pdf`.

Overview

TANGRAM has been designed as a general purpose tool but with a view to company data which are often best structured as multidimensional tables.

Basically, TANGRAM has been built around the idea of a *data cube*, that is, a multiway table containing both numeric cells and the labels that associate a meaning to every cell.

Every cube is physically represented by a TANGRAM database and stored on file.

The number of physical dimensions (or cube rank) in this version can range from 3 to 15.

In addition, the labels of each dimension can entail an unlimited number of fields (or *facets*) and, in this sense, the number of logical dimensions is unlimited.

The names of the dimensions are user-chosen (*for example MEASURES, PRODUCTS, MONTHS, CHANNELS, AREAS*).

Some of these dimensions deserve a special treatment (no matter what they have been named): for example the dimensions that represent *Variables, and Time periods.*

TANGRAM's approach to this is to reserve a special treatment to 3-way cubes in Standard Orientation, in which the dimensions represent Variables, Time periods and Items of some kind.

For instance, a string of cells along the dimension *Time periods* can be regarded as a *timeseries* and TANGRAM supports a good number of time-specific operations (like running sums, time lags, seasonal behavior,...).

Other fundamental operations (like computation, roll-up and selection), on the contrary, are meaningful on any axis.

In this manual, modules and keywords that only apply to cubes with three dimensions will be marked 3-way

Remember also that TANGRAM offers powerful techniques to change the cube rank and, more importantly, to move single facets from dimension to dimension or transform them into new dimensions.

Applications

TANGRAM has been applied in diverse fields, both to automate a single company function (for example: cashflow management, investment control, business plan, consolidation, ...) and to model the company as a whole.

The same TANGRAM databases that are created and managed for control purposes can become, without additional investment, the databases of an Executive Information System (EIS), that is to say, a software that allows the top management to freely navigate on company data and identify potential problems and opportunities.

♢ Although the creation and population of TANGRAM databases can be a matter of minutes, experience has shown that it is very recommendable to start with the preliminary operations of

- definition of the data structure, including data nomenclature and appropriate level of detail;
- data sourcing, that is, identification of the systems that will feed TANGRAM with relevant data (the main data source being typically the company information system).

It is neither necessary nor advisable to define all necessary outputs (reports, analyses, computations, ...) at the start, since TANGRAM is capable of obtaining nearly any output from its data structure.

Product profile

To conclude, TANGRAM's innovation can be summarized in the following points:

- support of a Cartesian database (rather than a relational one) which allows for global operations on multidimensional structures;

- a query and a computation language that were designed specifically for multidimensional structures;
- ease to grow upwards (TANGRAM procedures), downwards (programming) and side-ways (exchanging data with other software tools);
- seamless integration between ready-to-use modules, keyword languages and ad-hoc programming;
- safety and transparency, to ensure that only *meaningful* operations are permitted and every output is appropriately named.

If you wish to quickly spot TANGRAM's unique or innovative features, have a look at the sections on **CROSSOVER, BLOWUP, CONTRIBUTE, QUERY, GROWTREE, SHUFFLE, MERGE** in Data Analysis and also **BUILDDB** in Administration.

Caveat

Finally, where is TANGRAM applicable only *with caution* ?

1. In case you are happy with a spreadsheet. In case your data volumes are very reasonable, a two-way grid is sufficient and you don't see the need to handle hypercubes.

   TANGRAM's emphasis on 3-way cubes may lead you to think that the structure looks the same as rows, columns and sheets in a spreadsheet.

   In fact the comparison can go no further and the approaches are quite different. For example:

   - In a spreadsheet you want different formulae on a cell-by-cell level. TANGRAM is intended for safe mass operations, where you are unable to inspect your cells one by one.
   - The volumes you handle with TANGRAM are up to 2 Gigabytes - The maximum number of axes is 15.
   - A spreadsheet takes no responsibility for the meaning of data and does not care if you label your cells correctly or not. TANGRAM makes sure that a consistent set of labels follow your cells in all transformations.
   - TANGRAM emphasizes safe, documented, reproducible operations on very large and complex data structures.

2. In case your data structure is *profoundly relational*, meaning that very large element populations are mostly in a many-to-many relationship.

   TANGRAM uses a Cartesian data model, meaning that some or most populations are in an all-to-all relationship and become cube axes.

Typically, in real life, you have a mixed situation, where some relationships are many-to-many, some are many-to-one (children-parent), some are all-to-all.

TANGRAM handles many-to-many and many-to-one relationships as exceptions, using multi-facet labels on the cube dimensions.

In principle, all relational operations are possible on the labels of *each* dimension.

If, however, you end up with too many long facets on a dimension, you spend your time pointing at them with the mouse. If this becomes the rule, you might be happier with a record-and-field oriented tool or a relational database.

1.1 Basic terms

Axis See: Dimension

Cell A single number in an N-way cube. A cell can be identified by N indices or N element labels.

Co-ordinate See: Index

Cube An N-way numeric table plus N associated text labels that describe the dimension elements. This is TANGRAM's basic object.

Cube rank The number of dimensions of a cube. For instance, a 4-way cube has rank 4 or has 4 axes.

Database A cube that resides on file.

Dimension A population of elements on a cube axis and the associated text labels. For example: a list of Customers.

Dimension names Plural (collective) names of the elements of a dimension. For example: "Measures" "Months" "Salesmen"

Facet Some columns in a label that code an aspect of the elements and can be used like a field in a record. A facet is synonym of field, key, code, except that a facet is not formally declared in TANGRAM. For example: `2008 12 31` can be *regarded* as containing three facets. If all element labels have this structure `YYYY MM DD` then, say, the year facet can be used for sorting, selecting, joining, computing etc.

Flat file A fixed-field text file, typically used to import data into TANGRAM (or to export data). A flat file contains both codes, that become TANGRAM labels, and numbers (in char. format) that become cube cells.

Hypercube See: Cube

Indented namesystem (or indented label) A namesystem where initial underscores define an element aggregation tree.

Index The ordinal number that determines the position of an element within a dimension. For example: "Austria" has index 10 on dimension Countries.

Keyword A TANGRAM reserved name, typically used to provide an answer to a module (for instance: `ALL`) either alone or with other keywords (for instance: `ALL BUT 5 TO 12`).

Labels Texts that describe the elements of a dimensions. Each line is either free text, like `Profit before taxes` or is organized as fixed-length facets, like

```
CA LosAngeles   BDG 2007
IL Chicago      BDG 2006
```

Library A file that contains objects of three sorts: procedures, namesystems, user variables.

Module A TANGRAM self-contained program that typically transforms the current cube or performs an operation on a database.

Namesystems Reference labels that are not yet associated with a cube dimension.

Procedure (*alias*: macro, script). The sequence of calls that are issued in a TANGRAM worksession, saved in editable form.

Standard orientation A 3-way cube where the axes represent Variables, Time periods and Items of some sort. This is the simplest cube structure.

Time series A vector of cells on a time axis. If you keep all cube coordinates fixed and vary only the time coordinate, you obtain a time series.

User variable A variable assigned by the user (as opposed to those managed by TANGRAM).

Worksession All work you do with the TANGRAM modules (from the moment you leave the Main Menu).

1.2 Notation

Symbols in a box, like Ctrl ↑, represent keys to be pressed.

Commands in typewriter style, like `PLUS` , must be entered exactly as shown.

Text in italics, like *list*, will be replaced by the one of your choice.

For example, where the manual suggests the answer `ALL BUT` *list* the user might write `ALL BUT 3 5 29 31` .

Names in boldface, like **COMPUTE**, are self-standing modules that you find in menus and procedures. Keywords are shown in typewriter style, like `PICK` , and are typically used to provide a single answer to a module.

Paragraphs marked ♢ end an example or start a new topic.

The word Advanced marks advanced or rarely used features, or else details that mainly interest programmers.

The word 3-way marks modules or keywords that apply to 3-way cubes only.

The word *alias* is used to suggest that a term is also known under some other names. For example: a cube (*alias*: a database).

All TANGRAM names are case-sensitive. File names are not.

Chapter 2

Main menu and worksession

2.1 Overview

The TANGRAM menu system is structured as a tree.

When you start TANGRAM, (after the sunflower welcome screen) you get to the Main menu, which deals with procedure management, library management and initial TANGRAM configuration.

A *worksession* is defined as your activity in lower-level menus, that is, it begins when you choose Start worksession and ends when you return to the Main menu.

A *procedure* saves your last worksession for later use. Consequently, you cannot save the work done in the Main menu itsef, even if few modules of the Main menu (BROWSE, SELECTDB,...) duplicate those found in the worksession.

A *library file* is a collection of user objects: procedures, namesystems and private user variables.

A whole library is read in memory when you start, modified by your worksession activity and saved to file when you leave TANGRAM.

Libraries can be managed, copied, deleted with Windows Explorer but they are used only via TANGRAM services.

When you leave TANGRAM the current library is saved to disk, after confirmation. Current and secondary cubes (defined later in Data Analysis) are lost.

2.2 Main menu

♢ Menubar

- **File**
  - **Open database** — select the current database, losing the current cube

  - **New database** — create a new physical database
  - **Save cube as** — save the current cube as a new database
  - **Exit TANGRAM** — save the current library

- **Edit**
  - **Browse cube** — show the cell contents of the current cube
  - **Summary** — show the labels of the current cube
  - **Edit ASCII file** — edit a text file, possibly with framing characters

- **Procedures**
  - **Save worksession** — save the work done in the Session menu and below
  - **Update proc**
  - **Delete proc**
  - **Copy proc**
  - **List procs**
  - **Document procs**

- **Libraries**
  - **Save lib** — save the current library to file
  - **Save lib as** — save creating a new library
  - **Switch to lib** — the current libary is saved, then the new library is loaded
  - **Delete lib** — delete a library on disk
  - **Copy objects from** — selectively copy objects from a library on disk to the current library in memory

    After choosing from which library to copy, you are prompted to choose the objects to be copied (that will overwrite objects of the same name in memory).

    The prefix of the objects in the library listing helps you determine the type of object:
    * `proc` stands for a TANGRAM procedure;
    * `labl` stands for a label, alias a namesystem;
    * `user` stands for a private user variable.
  - **Clear lib** — delete all procedures, namesystems, private variables in memory

- **Configure**

- **Install password**
  This module completes the installation, defining the user, the password and the serial number.
  According to the password, your TANGRAM installation can be
  * Evaluation copy (the default). Database creation inhibited.
  * Single CPU license. All functions permitted but cannot be installed on a different PC.
  * Portable license. All functions permitted and, in case of need, the same parameters allow you to re-install TANGRAM.
- **Default dir**
  Advanced This module lets you define the default directories (*alias:* folders, paths) where user files are placed. This includes files of seven types: databases, libraries, masks, log files, HTML templates and outputs, LaTeX outputs, text files.
  Of course, you can place your files on any disk and in any directory, but it is tidy to keep all your files of the same nature together, so that TANGRAM may direct you to the applicable directory.
  If you change default directories, remember that this will take effect at once, so that TANGRAM will be at a loss to resume work without current database and library.
  To avoid this, it is a good idea to populate your new directories with a database and a library before you adopt them as your default.
- **Format chars**
  Advanced This module lets you redefine the formatting characters used by **REPORT** and **MASKPRINT**.
- **Miscellany**
  Advanced This module lets you define four global parameters:
  * Database lock.
    In a LAN environment, this is used to prevent conflicts when two users happen to access the same database.
    If you are using database sharing in a LAN, set this parameter to 1 in ALL involved PC.
    If you use 3-way TANGRAM on a single PC, set this parameter to 0, meaning that you don't need the lock.
  * Printed page (rows)
    The modules **REPORT, MASKPRINT** and **DRAFTPRINT** produce printed output with a fixed font. This parameter represents the number of printed lines per page in portrait orientation. The actual font size is computed according to the settings (resolution, orientation, ...) of the default printer.
  * Display only (rows)
    This is used to truncate very large tables after a given number of rows, to make the output more compact.

See, for example, the output of module **SUMMARY**.

∗ **BUILDDB** block (records)
The exploration of a flat file and the creation of a database are non-trivial and often time-consuming operations.
The performance of the module **BUILDDB** depends on the contents of the flat file (mostly on the way it is sorted) and also on the available RAM memory.
If you routinely import very large flat files from the mainframe, you may want to fine-tune this process to reduce the overall loading time.
The parameter represents the number of records that are read and processed at a time. Up to a point, performance is improved by increasing this parameter (at the risk of Memory Full errors).

- **Help**
  - **Manual** — open this manual `userman.pdf` in Acrobat reader
  - **Excel reader** — open the TANGRAM-Excel document `readerman.pdf` in Acrobat reader
  - **Installation readme** — open the installation notes in your browser
  - **TANGRAM homepage** — open the TANGRAM site in your browser
  - **About TANGRAM** — show the TANGRAM version

♢ Toolbar buttons

- **Worksession** — start the TANGRAM menus - start defining a procedure
- **Save worksession** — save your last worksession as a procedure
- **Update proce** — edit a procedure
- **Browse cube** — show the cell contents of the current cube
- **Manual** — open this manual `userman.pdf` in Acrobat reader
- **Exit TANGRAM** — Lose the current and secondary cubes, save the library
- **Arrange vert.** — Arrange the current labels on top of each other
- **Arrange horiz.** — Arrange the current labels side by side
- **Arrange slated** — Arrange the current labels in cascade
- **Select highlighted** — Select the highlighted elements - Reduce the current cube.
- **Reset database** — Select the whole current database

2.3 Session menu

- **Data Analysis**

  This menu contains all TANGRAM commands for exploiting databases to produce analyses, computations, printouts and many other outputs.

  Refer to Chapter 3 that explains these services in detail.

- **Administration**

  This menu groups several services related to the management of TANGRAM databases, for example, how to create one, define its labels, populate it with data, define formulae and so on.

  Refer to Chapter 4 that explains these services in detail.

- **Namesystem management**

  This menu contains all services for the management of namesystems (that is, reference nomenclatures) that are typically kept for later use on some cube dimension.

  For example: master lists of Customers, Products, Accounts,...

  Refer to Chapter 5 that explains these services in detail.

- **Run a procedure**

  This entry allows you to rerun a stored procedure. Since the procedure call is done *within* the worksession, this is a convenient way to create nested procedures.

  There is no preset limit to the nesting level.

  When a procedure is rerun, you may want to trace all steps (and perhaps change the answers) or you may want it to run unattended, in the background.

  Unattended execution is a good choice for production runs: it implies that the Questions and Answers box appears only in case of unacceptable answers, the Info box displays only warnings and errors and the Binary Choice box assumes the default reply.

  User options

  - the procedure name (in the current library)
  - execution mode (1 for step-by-step, 0 for unattended)

2.4 About procedures

A procedure keeps track of all modules successfully called in a worksession and the call arguments.

To create a procedure you simply execute the worksession, return to the Main menu and choose the option "Save last worksession".

The procedure name can be a character string of maximum length 25 characters, containing upper and lower case letters, digits and underscores '_'.

For example:

```
MONTHLY_REPORTS_2007
```

A procedure is permanently saved as part of the current library.

♢ When you execute a procedure, you may either confirm the same answers of the initial execution or modify them. You may also skip some modules by choosing "Cancel".

In this way, the re-execution produces *a clone* of the original procedure, which is automatically saved with the same name and a final underscore.

For example:

```
MONTHLY_REPORTS_2007_
```

These clones are not saved to the library and are overwritten at each new execution, so that it is your responsibility to rename them in case you want to keep them.

♢ It is a good habit to start a procedure with the choice of the database, so that it can be executed in any initial condition.

Procedures represent a simple and powerful way to "program" TANGRAM, producing safe and automatic applications for other users.

As noted, a procedure can call other procedures, with unlimited nesting.

A procedure can also call extra-menu TANGRAM programs or code of any type via the **EXECUTE** module in Data Analysis.

Procedures in the current library are read into memory at the start of a TANGRAM session and saved to the library before leaving TANGRAM.

More complex applications may want to split the collection of procedures in different libraries.

There is no practical limit to the size of the libraries or to the number and length of the procedures.

♢ A procedure contains a sequence of calls to TANGRAM modules, as illustrated in this example:

```
! Created on      18/07/2006 11:36:29 in library D:\new.LIB
! Last updated on
! Last run on
goSELECTDB 'D:\TANGRAM\HLM\msales.hlm'
goSUMMARY
goSELECT '2'  '5 10 15'
goCOMPUTE '2'  'QUARTER 1'  ' SUM [1 2 3]'
goNORM '2'  '4'
```

Advanced Beginners normally prefer to create procedures with a work-session —expert users may want to write and maintain procedures with the editor.

Editing a procedure, you may control the flow of execution with a control structure. The available keywords are documented in the Dyalog Help (Language help, Defined Fns&Ops, Control Structures) and exemplified here:

```
! Created on      12/02/2007 16:59:59 in library lib\modist.LIB
! Last updated on 14/02/2007 11:53:06
! Last run on     23/05/2007 19:03:54
!    Use of control structures in procedures
!    Test if old file exists:
:if ce 'TXT\TEMP1.TXT'
DOSERASE 'TXT\TEMP1.TXT'
5 INFO 'Deleted old TEMP1.TXT'
:else
5 INFO 'File TEMP1.TXT not found'
:end
:for MONTH :in 1 TO 12
goRUNPROC 'DO_MONTH' 'ON'
:end
:while X < Y
! Statement
! Statement
! Update X,Y
:end
```

A programmer may also add APL code, input-output arguments and local variables.

Advanced Each procedure line calls a program with the prefix `go` (a *frontend* version of the module).

Frontend programs will display their arguments during re-execution, let you modify them, check them for acceptability and save the call in the (clone) procedure.

Most programs also exist with the prefix `do` .

This *backend* version of the program accepts inputs in a more compact form, executes without user dialogue and does not check the acceptability of its inputs.

For example the TANGRAM module **SELECT** is implemented by a frontend and a backend programs. Typical calls may look like:

```
goSELECT  '1' 'ALL BUT LAST'
doSELECT   1  (1 TO 11)
```

Advanced Procedure executions are logged in file `log\proc.log` which can be useful to check the correctness of unattended executions. This is what the log file looks like:

```
22/01/2007 16:49:39 Sec.   428 ---- Start proc MONTHLY
22/01/2007 16:49:50 Sec.   439 ------- Start proc LOAD_MONTH
22/01/2007 16:50:23 Sec.   471 ------- End proc   LOAD_MONTH - sec.    33
22/01/2007 16:50:25 Sec.   474 ------- Start proc RECOMPUTE_ALL
22/01/2007 16:50:30 Sec.   479 ------- End proc   RECOMPUTE_ALL - sec.     5
22/01/2007 16:50:32 Sec.   481 ------- Start proc REPORTING
22/01/2007 16:50:38 Sec.   486 ---------- Start proc TO_BOARD
22/01/2007 16:50:42 Sec.   491 ---------- End proc   TO_BOARD - sec.     5
22/01/2007 16:50:44 Sec.   492 ---------- Start proc TO_STAFF
22/01/2007 16:50:50 Sec.   498 ---------- End proc   TO_STAFF - sec.     6
22/01/2007 16:50:51 Sec.   499 ------- End proc   REPORTING - sec.    19
22/01/2007 16:50:51 Sec.   500 ---- End proc   MONTHLY - sec.    72
```

Chapter 3

Data Analysis

Overview

This menu uses databases to produce analyses, queries, computations and printouts.

From this menu, you cannot alter the existing databases (this is reserved to Administrator services).

♢ At any time a current database and a current cube are defined. Optionally, you may also have a secondary cube.

Every command works on the current cube produced by the previous commands.

The commands **DELTA, MERGE, DISAGGREGATE, BLOWUP** require both current and secondary cube as their inputs.

Note that some commands are *executive* and alter the current cube, while others are just *informative* and produce output without modifying the cube.

♢ The current cube can be *virtual* if only the labels are in RAM memory, or *real* when the numeric cells are also in memory.

Some commands (like **SUMMARY, SELECT, RESET**) will work on virtual cubes, others will require the cube to be read into memory.

If your database is larger than the available RAM memory, you will have to work on a reasonably sized portion of it, on which all commands will be available. Typically you use **SELECT** on a dimension to reduce the cube size.

The traffic light before each command name will turn green when the size of the current cube is compatible with available memory.

The secondary cube (explained in this Chapter) is always real.

In some kind of computation, (**EXECUTE, GLOBAL, BROWSE**) you may reference the current and secondary cubes as `T` and `xT`

♢ The following sections present the available options, in menu order.

3.1 Virtual cube operations

3.1.1 SELECTDB

Function

Choose the current TANGRAM database.

The previous current cube is lost, the secondary cube is not.

The new current cube is initially equal to the whole database.

You can open only one database at a time.

Since the secondary cube is unaffected by this operation, it is an expedient way to assemble data from different databases. See "Secondary cube management" in this chapter.

User options

- the name of the database to open `.hlm`

3.1.2 SUMMARY

Function

Display the size and labels of current and secondary cube.

This is normally used to see the effect of previous commands before proceeding further.

Related topics

Topic	Section	Page
BROWSE	3.2.1	26
MISSING	3.2.3	34

3.1.3 SELECT

Function

Select, repeat or reorder the elements of a dimension.

Motivation

This command is typically used to reduce the current cube by selecting elements on one dimension at a time.

The same command will let you select some elements, repeat some and change their order.

The command works on each dimension independently.

♢ The power of this command is due to the number of available ways to express the element list.

You can express the element list as indices (like `3, 7, 10 TO 20, 7`), as labels (like `"Cost of goods" "Revenues" "Gross margin"`) or you can simply leave this input field empty to be prompted for an interactive choice.

A vast number of keywords, defined in the Set Language, let you express your choice in other ways, based on labels or cube contents.

For example your element selection may be:

```
WHERE ALLTRUE (T > 0) AND (T < 100)
```

Use the keyword `BUT` to code the elements to be dropped:

```
ALL BUT  13 TO 21
```

♢ The command **SELECT** is often used repeatedly on the same dimension. Remember, however, that the elements are renumbered each time.

After you select the element indices

```
5   4   6  12  10
```

they will be renumbered as

```
1   2   3   4   5
```

If you wish to keep track of the original indices, the `doLABELNUMBER` program (page 23) will write the numbering *inside* the labels.

Note that if you select less elements than you intended, you must start all over with a **RESET** command (and lose the selections on other dimensions).

♢ It is often useful to select the same element many times.

Example

If you keep the rate of exchange in a cube of *1 VARIABLE, 12 MONTHS, 1 COMPANY* you may want to expand it to *100 VARIABLES, 12 MONTHS, 24 COMPANIES* to make it conformable with the cube of values to be converted.

To achieve this, you simply select the Rate of Exchange cube on dimension 1 and supply this index list:

```
100 / 1
```

Then you select on dimension 3 with an index list

```
24 / 1
```

The notation `100 / 1` means the value 1 repeated a hundred times.

♢ Note that, by selecting with keywords, you may select zero elements. This is not reported as an error, but then you have no way to proceed further, except issuing a **RESET** command. Not all modules are guaranteed to correctly handle a cube with zero elements.

User options

- The dimension to select
- The index list, expressed in a variety of ways

Related topics

Topic	Section	Page
Set Language	6.2	127

3.1.4 RESET

Function

Reset the current cube to be equal to the whole database.

This command deletes the current cube and reassigns it as the whole current database. The current cube is initially virtual.

See the commands **SAVEAS** and **STORE** to save the current cube before issuing a **RESET**.

3.1.5 EXECUTE

Function

Direct code execution.

Motivation

This module provides a passthrough to the interpreter, letting you enter any command, expression or program name for immediate execution.

It can be used as a calculator or as an expedient way to call rarely used programs that are not included in the menu system.

Examples

You run **EXECUTE** and type in
```
1234 + 4567
```
In the output window you get
```
5801
```
To force zeroes where the current cube has missing values:
```
ND FORCE 0
```
To transform the cube values into rank positions 3-way :
```
doRANK
```

Advanced **EXECUTE** expects a result and most programs produce a result or a return code. A program that produces no result is best preceded by

the ♢ symbol. Otherwise the program may run correctly but the user will be warned of a `Yields no result` error.

Advanced As mentioned, the **EXECUTE** command is a general purpose passthrough to the APL interpreter. There is no preset limit to the kind of instructions that you can submit (at your own risk) via this trapdoor. See the section The APL interpreter on page 140 for minor differences between standard APL and what **EXECUTE** accepts.

Advanced You can assign private variables with a colon ′:′ symbol.

If the variable name begins with `user` then the variable will be saved in the current library (Note for programmers: nested arrays are not supported in libraries). Otherwise the variable is lost when you leave TANGRAM.

Examples:

```
PI : 3.14159
XYZ : COUNT 3
userNAME : 'Dept. of Agriculture'
```

In rare cases there might be a conflict between your variables and TANGRAM's names. This is avoided if you use names with the prefix `user`

User options

- A program name or, in general, an instruction to execute.

Errors and warnings

- Any interpreter-level error produced by your instruction. For example:

```
Yields no result
Syntax error
Illegal argument
Illegal index
Length mismatch
System limit error
Wrong rank
Memory full
```

3.1.6 Label manipulation

Function

Manage the labels of the current cube (not in the database).

Motivation

The labels are codes or texts that describe the elements of each dimension. They are a crucial part of database definition and are best chosen as compact, informative names and codes that optimize data management.

In particular you may have multi-facet labels to classify the same element population according to several aspects.

These commands allows you to alter these labels at a later phase, so that they become suitable for displays and printouts.

Example

If you manage a group of companies, you might want to classify each company with three facets (state, size, business field), like in this example:

```
PEGASUS Ass.          TX   L   HW
ACME Corp.            MI   XL  SW
JOHN DOE & C.         CA   S   SW
```

These additional facets provide many-to-one associations that can be exploited to select, aggregate, sort, ... the elements.

Later, you may want to delete these codes or replace them with full descriptions, before the cube is ready for printing.

♢ In addition to the menu modules, these backend modules are also relevant to label manipulation (use them via **EXECUTE**):

- `doLABELAXES` *string1 string2 string3 ...*

  Assign new names to the dimension names (of the current cube). Empty strings leave the name unchanged.

  For instance:

  ```
  doLABELAXES   ' ' 'Months/Versions' 'Keys'
  ```

- `doLABELGROW` *dimension text*

  Append text to the current cube labels.

  This utility program appends the char string *text* to all labels of the current cube on a given *dimension*.

  For instance:

  ```
  doLABELGROW  3  ' Forecast'
  ```

- `doLABELNUMBER` *dim*

  Append ordinal numbers to the labels of dimension *dim*.

  Sequence numbers (like `#001 #002 #003` ...) are inserted inside the labels.

  This can be useful if you want to trace the initial element position during a sequence of **SELECT, COMPUTE** and **CROSSOVER** commands.

  In other cases, you end up with duplicates in the labels and want a way to differenciate them.

- `doLOCATE` *dim search-string*

  This command will assist you in locating a label in a large population.

  You can supply the beginning of the first words and obtain a list of hits, that is, labels that match your request.

  The search is case insensitive and ignores accidental differences such as accented vowels, punctuation signs, multiple spaces etc.

  Example

  A label such as

  `Depreciation fund (ex art. 26 D.P.R. 12.2.87)`

  can be located with one of these calls

  `doLOCATE 1 'DEP FUND EX ART 26'`

  `doLOCATE 1 '  D   F  EX '`

  `doLOCATE 1 ' dep fund'`

3.1.6.1 LABELCOLUMNS

Function

Select, drop or reorder the label columns.

You normally choose the dimension and leave the selection mask empty, so that a Column Selection Box is opened for a convenient interactive choice.

If you save a procedure containing this choice, you will notice that the selection mask is stored as a character string, like `'2221111333'`.

User options

- the dimension
- the column selection mask

3.1.6.2 LABELJOIN

Function

Replace a facet in the labels with its full description taken from a namesystem.

This is known as a *join* operation and requires a common key in the label and namesystem.

You point to the key position both in the label and in the namesystem and you get the namesystem contents inside the labels.

Example

You start with labels containing customer codes:

```
002
003
005
009
```

You reference a namesystem `CUSTOMERS` having a full description of the customers:

```
001 TX Hewlett Packard
002 CA Synaptics
003 MA Digital
004 VA Orion
005 IL Pegasus
```

The key that drives the join operation is, of course, represented by the first three characters in each table (this is the *primary key* in the namesystem).

After the operation, the current cube labels will display *both* old and new contents:

```
002|002 CA Synaptics
003|003 MA Digital
005|005 IL Pegasus
009|???????????????
```

Typically, this operation is followed by the **LABELCOLUMNS** command, to select the columns to be retained.

User options

- the dimension
- the namesystem
- the label mask
- the namesystem mask

Errors and warnings

- Warning - Duplicate elements in the namesystem
- Warning - Some codes were not found in the namesystem (they are replaced by a string of `'???'`)

3.2 Cube inspection

3.2.1 BROWSE

Function

Cube display, update, copy to clipboard.

Motivation

This module opens a spreadsheet-like window, letting you browse (or update) the current cube, print it selectively and copy individual pages to the Windows clipboard.

Two cube dimensions will be expanded as rows and columns. The rest will become combos that allow you to move on each axis. Remember that you are displaying and updating the current cube, not the database.

You may also use a color code to classify the cells according to a given criterion. For example, you may want to highlight negative values, values below budget, actual versus forecast etc.

The definition of updated and read-only cells can be done with a single value for the whole cube, but also with a formula.

Both formulae (cell color and update rights) obey the general rules of the **GLOBAL** command, with the addition of specialized keywords which are listed in this section.

User options

- Choice of the cube dimensions shown as rows and columns
- Formula that defines colors (in the range 0 to 7) on a cell-by-cell basis
- Update rights on a cell-by-cell basis.

3.2.1.1 Formula syntax

If you are not interested in coloring, leave the default formula (`green`).

Any formula that produces integer values in the range 0 to 7 is acceptable.

The names of the colors are

```
black, blue, green, cyan, red, magenta, yellow, white
```

Several specific keywords are available:

- *color-list* `UPTO` *threshold-list*

  Define the color based on the value in the cell.

  N increasing thresholds define $N+1$ intervals, to which $N+1$ colors apply.

  Example

  Negative values must be green, values in the range 0-100 red, values above 100 white:

  ```
  (green,red,white) UPTO  0 100
  ```

- *color-list* `percentile` *threshold-list*

  3-way Define the color based on the population distribution. (For instance, best 5% and worst 5%)

  A population is defined on dimension 3 and percentiles are computed on this population, according to your threshold list.

  N increasing thresholds define $N+1$ intervals to which $N+1$ colors apply.

  Example

  Your current cube entails 3 Ratios, 5 Years, 100 Companies. For each ratio-year pair you have a population of 100 Companies in which you want to detect the best and worst 5% performance.

  The best 5% must be yellow, the worst 5% red, the rest green:

  ```
  (yellow,green,red) percentile  5 95
  ```

- *dimension* `INLIST` *list*

  Define the color based on the cell index along a dimension.

  For example, `2 INLIST 3 6 9 12` refers to elements 3,6,9,12 on dimension 2 of the current cube (irrespective of your view, that is, row and column choice).

  This keyword returns a Boolean table of the same size as the current cube. This table is typically used within an IF-ELSE construct.

  Example

  Periods 3, 6, 9, 12 are end-of-quarter months and must be shown in white, the other periods in green:

  ```
  (white IF 2 INLIST 3 6 9 12)
     ELSE (green IF 2 INLIST 1 2 4 5 7 8 10 11)
  ```

  Since green is the default color, in this example the ELSE part may not be coded:

  ```
  white IF 2 INLIST 3 6 9 12
  ```

- `SUB`

  Define a cube subset as the intersection of element sets on all dimensions.

  This keyword will ask you to select a subpopulation on each cube axis and will return a Boolean table of the size of the current cube.

At the intersection of seleted subsets, the result will be 1; elsewhere it will be 0.

Example

The area selected by `SUB` must be yellow, the rest red:

```
(yellow IF SUB) OrElse red
```

If you code several calls to `SUB` in the same reply, you should be aware that the order of execution is right to left, so that the first set of choices will answer the rightmost `SUB`:

```
(yellow IF SUB) OrElse  (red IF SUB)
```

♢ This is an open-ended language in which you can freely mix computations, Boolean operators, variable names and keywords.

If you happen to have a secondary cube of the same shape as the current one, you refer to them as `xT` and `T` respectively.

Examples

```
red IF T > 1320

red IF (T > xT TIMES 1.05 ) OR (T < xT TIMES 0.95)

white IF (T = 0) AND NOT (xT = 0)

yellow IF (norm T) > 0.02
```

The last example will display in yellow all cells above 2% of the total cube, irrespective of their coordinates.

Advanced Color codes disappear when you quit **BROWSE** (although you may save the call and apply it to a different cube). But if you want color codes to represent cell attributes that are saved with the cube, TANGRAM doesn't support you.

Or, does it? With little extra effort you can have cell values and cell attributes faring together and saved with the cube.

See the main steps:

1. **STORE** the current cube
2. Call **BROWSE** and assign your color formula to `xT` , for ex.

   ```
   xT: (red IF T<0) OrElse white
   ```
3. **PROMOTE** the current cube with a new element "Value"
4. **SWAP**

5. **PROMOTE** the current cube with a new element "Attribute"
6. **STORE** the current cube on the last dimension
7. **GETBACK**
8. **SAVEAS**

The net result is a cube with an extra 2-element dimension, which can be updated via data entry. Your attributes may code for any permanent cell property, like: 0=estimate 1=forecast 2=actual 3=external source etc.

3.2.1.2 Update rights

If you are not interested in cell-by-cell update rights, use a global reply, like `ON` or `OFF` (which stand for 1 and 0).

Otherwise, all keywords and techniques that were proposed for the color formula apply here also.

The only difference is that you are defining Boolean values, not integers 0 to 7, and that in this case the default is 0.

Examples

Let me update only the cells with a missing value:

```
ON IF T = ND
```

Let me update periods from September to December:

```
ON IF 2 INLIST 9 TO 12
```

Let me define a subcube to update and also update negative cells everywhere:

```
ON IF SUB OR T < 0
```

Advanced Let me define the update rights on the result of the color formula, so that I reply to `SUB` only once. This is accomplished by assigning a result to a variable. The color formula may be:

```
ABC:  (red IF SUB OR T= xT) OrElse white
```

and the update rights may refer to ABC:

```
ON IF ABC=white
```

3.2.2 QUERY

Function

3-way Formulate an English-like query on the current cube. This applies to 3-way cubes in Standard Orientation, with axes meaning Variables, Periods, Items.

Motivation

This module offers access to a sophisticated query language.

It is understood that you have Variables on dimension 1, Periods on dimension 2 and Items on dimension 3 (The actual dimension names are unimportant—they might be called Measures, Months and Customers).

The query is formulated on Variables and returns a collection of Items.

The object of each query is then a single time slice (Variables and Items) and the query is simply iterated on each period.

The query syntax is designed for good readability: a query looks like a natural-language sentence that uses a limited set of keywords.

This is intended to approach an ideal situation where:

- sentences that sound acceptable in English are also correct queries;
- the query produces what one would expect by the English meaning.

In more practical terms, a query consists of Variable names and keywords from the following set:

```
count average total max min
equal from to above below
where label and or not
```

Variable names can be expressed in the double-quote notation, that is, they can be shortened and enclosed between `"  "`.

If, for example, your labels are:

```
Revenues from sale
Revenues from services
Revenues (other)
```

then the notation `"Revenue"` will be rejected as non-unique, `"Revenues from sales"` will be rejected as unknown, while `"Revenues sale"` will be accepted as unique.

As mentioned, a query will let you select Items with Boolean conditions of any complexity (linked by `and, or, not` operations), extract values or labels of these Items and apply summary operations (`count, average, total, max, min`) to the selected set.

Examples

The examples apply to a hypothetical cube containing *VARIABLES, YEARS, COMPANIES.*

```
"EMPLOYEES" where "NUMBER OF BRANCHES" above 10

average "MARGIN" where "TOT. REVENUE" from 10000 to 30000
```

```
count where not "DEPRECIATION FUND" below 900000

label where ("CAPITAL" from 100000 to 200000) or
("REVENUES" above 100000)

"ROI" where "CAPITAL" from 0 to 50000

("COSTS", "REVENUES", "FIXED ASSETS") where
("REVENUES/CAPITAL" above 55.5) or
("COSTS/CAPITAL" below 15)

max "ROI"

average "INVESTMENTS/TURNOVER" where not
"TURNOVER" from 2000000 to 5000000

label where ("EMPLOYEES" above 500) and
("NUMBER OF BRANCHES" above 250) and ("COSTS" above 100000)
```

As it is the case with the **COMPUTE** command, Variables can also be identified by their indices.

The same query examples may be formulated in a more compact form:

```
[1] where [2] above 10

average [3] where [4] from 10000 to 30000

count where not [5] below 900000

label where ([5] from 100000 to 200000) or ([6] above 100000)

[5] where [7] from 0 to 50000

([8], [9], [7]) where ([10] above 55.5) or ([11] below 15)

max [12]

average [13] where not [6] from 2000000 to 5000000

label where ([14] above 500) and ([2] above 250)
and ([15] above 100000)
```

It goes without saying that the labels that appear in a query are the labels of the current cube, not necessarily those in the physical database.

Similarly, the values in the current cube are the result of all previous operations, so that a query may concern percent variations, normalized values, ratios and what not.

Queries that begin with `where` return the element indices (for example: *2 7 11*) whereas queries that begin with `label where` return the corresponding labels (for example: *'FIAT', 'VOLKSWAGEN', 'OPEL'*).

This is another open-ended language that can be freely mixed with all keywords available in TANGRAM. You find a formal definition of the syntax of a query in the figures at the end of the manual.

♢ Advanced

QUERY and missing values.

Suppose you have a complete 3-way cube in which three variables should add up to 100. To spot items where exceed 100, you may issue a query like:

```
label where 100 < [1] + [2] + [3]
```

If you do have missing values, the query must be formulated with keywords that handle them:

```
label where  100 below [1] PLUS [2] PLUS [3]
```

This will correctly show all elements where none of `[1] [2] [3]` is missing and the sum is not 100.

The keywords `equal above below from to` handle missing values and return a Boolean one where their arguments are not missing and satisfy the inequality. Missing values are left out.

If you have 1000 elements on dim. 3 and issue the queries

```
count where [1] below 500
count where [1] equal 500
count where [1] above 500
```

you may be puzzled to find that the results are 720 0 240 respectively. The 40 missing cases are items where variable `[1]` is missing. To count these you must use `=` and not `equal` :

```
count where [1] = ND
```

To wrap up, extra care is needed with missing values. While **QUERY** keywords can only be used in a query, computational keywords

```
PLUS MINUS TIMES DIVIDE INT MOD ROUND APPROX
```

can be used in any context, including a query.

User options

- The query
- Output destination (screen, printer, file)

Related topics

Topic	Section	Page
`WHERE` keyword	6.2	127
RANGE	3.2.4	34

3.2.3 MISSING

Function

This command reports on the positions of missing values in the current cube.

Motivation

A missing value (*alias*: null, empty cell) may represent either a value that has never been entered or the result of a computation that involved missing values (for example the result of `PLUS, MINUS, TIMES, DIVIDE, PRE, POST`).

The module will report on the total number of missing values and, for each dimension, display the list of:

- complete elements
- incomplete elements
- empty elements.

3.2.4 RANGE

Function

3-way (Variables, Periods, Items)
Classify the elements of dimension 3 according to cell values.

Motivation

This module grades or classifies the Items (elements on dimension 3) according to the values in the corresponding cells.

You specify the Variable and the Period to be used for the operation and the thresholds or class borders.

Example

You wish to classify your customers according to the volume of business.

Your classes may be: nil—100, 100—500, 500—1000, 1000—up

Your class borders are then expressed as `100 500 1000` and define four intervals.

You also have to decide which variable to use (say, Net Sales) and which period (say, Year 2006).

After the operation, a new field is appended to the customer labels, containing one of the four classifications:

```
... -- 100
100 -- 500
500 --1000
1000--...
```

In case you don't know the order of magnitude of the values involved you can use the keyword `DEFAULT` instead of the class borders. (Reasonable class borders will be obtained on the basis of the actual values).

User options

- Element on dimension 1
- Element on dimension 2
- Class borders (a string of increasing values or the keyword `DEFAULT`)

Side effects

- The labels of dimension 3 are updated.

Related topics

Topic	Section	Page
QUERY	3.2.2	30
`WHERE` keyword	6.2	127
`PERCENTILE` keyword	3.3.1.1	39
`percentile` keyword	3.2.1	26
`MAJOR` keyword	6.2	127

3.3 Computations

3.3.1 COMPUTE

Function

Compute a new element and append it to the current cube.

Motivation

This is the general purpose tool for computing.

With a few exceptions (see following sections), it works on a dimension at a time.

If, for example, your application concerns *ACCOUNTS, YEARS, COMPANIES*, the command will create a new account or a new year or else a new company.

If you create a new account, it will exist for all years and all companies.

♢ The notation that defines the computation is an open-ended language that ranges from elementary to very sophisticated.

The rules for coding are:

- the formula is written in free format, of any length.

- the formula may contain keywords (`PLUS, MINUS`...), element indices (`[39]`), labels (`"fixed overhead"`), parentheses and numeric constants. The keywords are listed in a later section.

- within parenthesis there is no priority between keywords (they are executed right-to-left).

- The double quote notation must single out an element: one and only one label must contain the words within the double quotes. (Case-sensitive, position-independent search).

Example 1

To sum elements *10* and *15*, you enter the formula:

```
[10]  PLUS  [15]
```

Example 2

To compute the sum of elements *10* and *12* less elements *14, 15, 16*:

```
( [10] PLUS  [12] ) MINUS ( [14] PLUS [15] PLUS [16] )
```

Example 3

```
"Revenue from sales" DIVIDE "Total Revenues"
```

This formula will be accepted if one of the labels contains the three strings `Revenue from` and `sales` (in any order) and a second unique label similarly contains the strings `Total` and `Revenues`.

If such elements happen to be the 23rd and 71st, then the formula is equivalent to:

```
[23] DIVIDE [71]
```

The double quote notation can be freely mixed with the square brackets.

The maintenance of a procedure is normally more convenient if elements are referenced by label rather than by index.

♢ If you have long lists of elements to sum, the keyword `SUM[` *list* `]` represents a convenient short-hand notation:

Example 4

```
SUM [ ALL ]
SUM [3 4 10 TO 19 ]
```

```
SUM [ SPOT  'Savings Bank']
```

♢ Keywords like `PLUS, MINUS, TIMES, DIVIDE` are just synonyms of the arithmetic operations, but with a difference: they handle missing values in the result.

In other words the expression:

```
[3] PLUS [18]
```

will create a new element with missing values where the 3rd or 18th elements exhibit missing values.

The `DIVIDE` keyword will also produce a missing value where the right argument is zero.

Example 5

The ratio between these two incomplete timeseries (with ND marking a missing value)

```
40   20   ND   ND   60   100   ND   ND

 5    5   ND    0    0    10   10    8
```

will produce the following timeseries

```
 8    4   ND   ND   ND    10   ND   ND
```

User options

- the dimension along which to compute
- the label of the new element
- the formula that computes the new element as a function of existing elements.

Errors and warnings

- Error - An empty label is not allowed
- Error - Mismatched parentheses
- Error - Reference to non-existing elements
- Error - Illegal use of the double quote notation
- Error - Result has unacceptable size
- Error - Formula execution failed

Related topics

Topic	Section	Page
UPDFORMULAE	4.4.1	106
GLOBAL	3.3.4	47
CONTRIBUTE	3.3.6	49

Contrast the **COMPUTE** command with similar facilities of the Administration Menu:

- Administrator formulae are used to define computations of general interest, for the benefit of all users of a database; they are saved in the database and managed by TANGRAM.
- Administrator formulae are applied, as a rule, to the whole database. The formulae supplied to **COMPUTE** are 'private' in the sense that they are defined and saved by a user and do not update the database. They are typically saved in a procedure.

3.3.1.1 Computation keywords

The keywords that can be used in a **COMPUTE** formula are summarized in the following table (with the exception of the IF—ELSE constructs).

Syntax	Meaning	Example
x `PLUS` *y*	Sum	`[2] PLUS [3]`
x `MINUS` *y*	Difference	`[2] MINUS [3]`
`MINUS` *y*	Negative of y	`MINUS [3]`
x `TIMES` *y*	Product	`[2] TIMES [3]`
x `DIVIDE` *y*	Division	`[2] DIVIDE [3]`
`DIVIDE` *y*	Inverse of y	`DIVIDE [3]`
x `LESS` *y*	Difference	`[1] LESS [3]`
x `OVER` *y*	Division	`[2] OVER [1]`
`INT` *x*	Integer part of x	`INT [3]`
`MOD` *x*	Absolute value of x	`MOD [3]`
x `ROUND` *y*	Round y to a multiple of x	`1000 ROUND [3]`
`ROUND` *y*	Round to the nearest integer	`ROUND [3]`
`SUM` *list*	Sum of a list of elements	`SUM [3 TO 20]`
`PRODUCT` *list*	Product of a list of elements	`PRODUCT [3 5 9]`
`PRE` *x*	The previous element (time shift)	`PRE [7]`
`POST` *x*	The next element (time shift)	`POST [7]`
n `SLIDE` *x*	Timeshift of n periods	`12 SLIDE [7]`
`todate` *x*	Running sum (on time axis)	`todate [7]`
`toperiod` *x*	Diff. from prev. period	`toperiod [7]`
`norm` *x*	Normalize x so that it sums to 1	`norm [7]`
x `OrElse` *y*	Take y where x has missing values	`[1] OrElse [7]`
`ND`	No Data. A single missing value	`ND, 100, ND, -5`
`STAT` *n*	Statistics on a population 3-way n=1 the mean n=2 the variance n=3 the third-order moment, etc.	`STAT 1`
`PERCENTILE` *n*	Compute the n-th percentile 3-way	`PERCENTILE 95`

These keywords have been designed to correctly handle missing values.

If the current cube is complete (free from missing values) one can use the traditional symbols with the same effect (for example, `+` instead of `PLUS`).

The set of available primitive symbols of the interpreter is very vast and includes exponentiation, logarithms, trigonometrical functions and much more.

Finally, remember that element collections in a formula may be expressed with the Set Language.

Examples

```
"Revenues"  LESS  "Overhead costs"
```

```
PRODUCT [ ALL BUT 19 ]
```

3.3.1.2 IF-ELSE conditions

Advanced This section defines an extension of the computation language that allows for conditions on all dimensions.

If *exp* is a valid formula and *cond* a Boolean condition, the extension is of the general form:

((*exp1*) `IF` *cond1*) `ELSE` ((*exp2*) `IF` *cond2*)
... `ELSE` ((*expn*) `IF` *condn*)

The meaning is to execute *exp1* if *cond1* is satisfied, *exp2* if *cond2* is satisfied and so on.

The total length of the formula is unlimited. The number of `IF` conditions is also unlimited.

A condition is any expression that produces a Boolean result (either 0 or 1).

As a general rule, it is best to use conditions that are mutually exclusive and cover all possible outcomes, for example:

cond1: T < 0
cond2: T $= 0$
cond3: T > 0

Example 1

A conditional formula for **COMPUTE** that satisfies these requirements:

```
([1] IF [3]<0) ELSE (0 IF [3]=0) ELSE (([1] DIVIDE [3]) IF [3]>0)
```

♢ The typical use of conditions is to make the formula dependent on elements of the other dimensions. This is done with the keyword

dim `INLIST` *list*

which return a Boolean 1 for cells that belong to the element *list* on dimension *dim*.

Example 2

If you compute on dimension 1 you may want to use a different formula for periods 1 and 3:

```
([1] IF 2 INLIST 1 3) ELSE (([1] DIVIDE [3]) IF 2 INLIST 2 4 5)
```

♢ TANGRAM will protect you from several mistakes and limiting cases (by assigning missing values where appropriate):

- cases in which no condition is verified
- cases where several conditions are verified

- ill-formulated conditions that produce non-Boolean results
- use of `INLIST` with illegal list of elements.

Syntax errors will produce an error warning and prompt you to re-enter a correct formula.

Example 3

If you are computing on *Variables* with the formula:

```
(100 IF 2 INLIST 2) ELSE (200 IF 3 INLIST 1)
```

the result will be a new *Variable* with the constant value *100* in the second period, the value *200* on the first item and missing values elsewhere (including the intersection of second period and first item).

Example 4

When computing on dimension 1:

```
(([7] DIVIDE [6]) IF 2 INLIST 3 6 9 12)
ELSE (0 IF 2 INLIST 1 2 4 5 7 8 10 11)
```

Example 5

When computing on dimension 2:

```
(([12] TIMES 1.175) IF 1 INLIST  3 TO 11, 18 20)
ELSE ((0.5 TIMES [11] PLUS [12]) IF 1 INLIST  2 19)
ELSE (0 IF 1 INLIST  1, 12 TO 17)
```

Example 6

When computing on dimension 3:

```
([2] PLUS [7]) IF 1 INLIST 10 12
```

Example 7

When computing on dimension 3:

```
(SUM [ALL]) IF 2 INLIST (13 TO 24) BUT 18
```

♢ The keyword `OrElse` is handy to assign a default formula where a chain of IFs has failed to cover all possibilities.

Formally, `OrElse` assigns cells of its right argument only where the left arguments has missing values.

Example 8

When computing on dimension 2:

```
( (SUM [ALL]) IF (3 INLIST 3 6 9) OR 1 INLIST 2 4)  OrElse 0
```

3.3.1.3 Time shifts

Advanced The plain use of **COMPUTE** implies a separation between the dimensions: if we apply the command to a dimension called *ACCOUNTS*, the formula may reference all existing accounts but no other dimensions.

This section presents an exception to the basic rule: when you compute on the *first* or *third* dimension of a 3-way cube if you may refer to elements of the previous or following time periods.

Remember that in Standard Orientation the meaning of the dimensions is *Variables*, *Periods*, *Items*.

Time lags are expressed with the keywords `PRE` , `POST` and `SLIDE`.

Example 1

To compute the average of variable *12* between a period and the previous one :

```
([12] PLUS PRE [12]) DIVIDE 2
```

Example 2

When you compute on *ITEMS*, the formula

```
[1] PLUS (POST [1]) PLUS PRE [1]
```

will yield the sum of item *1* on the current period, the previous one and the next one (lagging and leading values).

Note the use of parentheses to force the order of execution.

♢ The keywords `PRE` and `POST` will produce zeroes at one end of a timeseries, where the previous or next periods do not exist.

Note that the keywords `PRE` and `POST` do not apply to computation on the second dimension.

If you are computing on periods, the notation `[12]` stands for the 12th period and the previous period is simply denoted as `[11]`.

♢ There is no restriction on the repeated use of `PRE` and `POST`, for example:

```
POST POST POST [10]
```

The keyword `SLIDE` is convenient to avoid this repetition, for example:

```
3 SLIDE [10]
```

Positive values of the left argument of `SLIDE` represent shifts to the future.

Note that `SLIDE` will also produce zeroes at one end of a timeseries.

Advanced The keywords `toperiod` and `todate` are also relevant to time shifts.

Their function is similar to that of **TOPERIOD** and **TODATE**, in that they transform to-date values to period values and viceversa, but they apply to a single element, not to the whole cube.

Example

You have a variable `Invoices` (on dim. 1) that keeps track of the invoices received on a monthly basis. Suppose the terms of payments are 3 months, so that each invoice will produce a debt for the duration of 3 months.

You can compute a new variable `Current debt` (on dim. 1) with the formula

```
todate "Invoices" MINUS 3 SLIDE "Invoices"
```

♢ Time shift keywords handle missing values and also apply to N-way cubes, assuming that the second dimension is a time axis. Since their use in non-trivial, it is recommended that you learn their use on a 3-way cube to start with.

3.3.2 ROLLUP

Function

Roll up (*alias*: aggregate or group) the elements of a dimension.

Motivation

This module brings the current cube to a higher level of synthesis by summing the elements according to a key.

The **ROLLUP** command works on one dimension at a time.

You simply point to the label facets that you want to get rid of: *those elements that differ only on those positions will be added together to form a single element.*

It is your responsibility to see that the the variables invoved may be meaningfully added (interest rate and percent growth, for example, may not).

Aggregated positions in the labels are marked with `+++` signs, to remind you of their new meaning.

Missing elements are aggregated as zeroes.

Example 1

Let your dimension *Months* contain the following labels:

```
2007 Q1 JAN
2007 Q1 FEB
2007 Q1 MAR
```

```
2007 Q2 APR
2007 Q2 MAY
2007 Q2 JUN
2007 Q3 JUL
```

This label structure lets you roll up from monthly data to quarterly and yearly data by choosing the appropriate key.

If you choose an aggregation key like

```
111
```

then you roll up from months to quarters.

After aggregation, a new label, say

```
2007 Q1 +++
```

will contain the sum of the original elements

```
2007 Q1 JAN
2007 Q1 FEB
2007 Q1 MAR
```

A second execution of **ROLLUP** on the same dimension may yield the total, which will look like

```
2007 ++ +++
```

♢ If you use this kind of multi-facet labels to represent a hierarchy on several dimensions, you can, of course, "climb the ladder" in many steps, to see all intermediate aggregations.

A single hierarchy on each dimension is the most common situation. The key structure is, however, more general: the various facets can represent several aggregation paths starting from the same atomic elements.

Example 2

If you have a collection of *BANKS*, you may classify them by size, by bank type and by region.

Your labels might look like

```
LRG  SAV  CH    Lausanne Savings Bank
SML  MRC  IRL   Tipperary Merchant Bank
```

and will let you get size- type- and region- totals.

♢ If you have a complex multi-level hierarchy and want to count, say, the number of branches in each state or in each region, you simply create a new variable equal to 1 everywhere (via **COMPUTE**) and then roll up to the desired level, aggregating both significant variables and your tally.

With a similar technique you can obtain averages, instead of sums.

User options

- Dimension to roll-up
- Roll-up key (label positions that must disappear in the process)

Side effects

- The labels of the chosen dimension are updated.
- The current cube is recomputed.
- Missing values disappear.

Related topics

Topic	Section	Page
BLOWUP	3.4.4	56
COMPUTE	3.3.1	35
DISAGGREGATE	3.4.5	57
BUILDDB	4.3.4	100
GROWTREE	3.3.3	45

Note. The module **BUILDDB** lets you aggregate values in a flat file while the file is transformed into a TANGRAM cube.

3.3.3 GROWTREE

Function

Instate on a dimension an indented namesystem that defines an aggregation tree. Recompute the cube accordingly.

Motivation

While the **ROLLUP** module aggregates a facet, **GROWTREE** offers a custom aggregation tree, with any number of steps and a variable depth on each tree branch. It assumes you have an indented namesystem in which the leaf elements are the labels of the current cube and computes the upper levels.

Labels that are not in the indented namesystem are lost. Leaf elements in the namesystem that are not in the current cube labels are set to zero.

Example

Assume that your labels represent product codes:

```
003
005
001
007
```

GROWTREE will install a namesystem like this

```
All products
___Modems
______Internal
_________001
_________002
_________003
______External
_________004
_________005
_________006
_________007
___Hard disks
______009
```

The number of underscores for each step is free (3 in this example). Codes like `004` are leaf elements since they are not followed by a deeper indentation.

Some totals, like `___Hard disks` will be set to zero by **GROWTREE**, as the original cube does not have any relevant data.

Advanced The command **GROWTREE** selects, excludes, reorders, repeats elements, taking care of various extreme cases:

- The current cube may not have duplicate elements.
- The namesystem may have duplicate elements of any kind, without ambiguity. If a tree branch is equal to a leaf element, it will be computed according to the hierarchy.
- If a code in the tree is followed by a description, it will be correctly assigned, since comparison is done on the initial label width.
- The indented namesystem may contain several hierarchies of the same atomic elements, appended to each other. For instance, the same four codes of our example

  ```
  003
  005
  001
  007
  ```

  may be expanded into

  ```
  All products
  _USB port
  __003
  __005
  ```

```
_Serial port
__001
__007
All products
_Business
__005
__001
_Home
__Outdoors
___007
__003
```

User options

- Dimension that grows
- Indented namesystem

Side effects

- The current cube is updated
- Missing values become zeroes

Related topics

Topic	Section	Page
ROLLUP	3.3.2	43
UPDFORMULAE	4.4.1	106

3.3.4 GLOBAL

Function

Recompute all cells of the current cube, without altering the labels.

Motivation

While **COMPUTE** appends a computed element on a single dimension, **GOLBAL** recomputes all cells.

The **GLOBAL** formula may use most keywords available to **COMPUTE**, with the exception of

```
SUM PRODUCT todate toperiod STAT PERCENTILE
```

The formula may not refer to single elements (with a notation such as `[12]` or `"Overhead"`).

Instead, the formula refers to the current cube as `T` and, if applicable, to the secondary cube as `xT` .

♢ While in general your formula applies to the whole cube, the keywords `SUB` and `INLIST` provide a way to limit the scope of the formula to a subcube.

`SUB` will ask you to define a subcube by selecting elements on all axes, while `INLIST` specifies a list of elements on a single axis. These keywords are also explained under **BROWSE** in section 3.2.1 on page 26.

The keywords `IF, ELSE, OrElse` are explained under **COMPUTE** in section 3.3.1 on page 35.

Examples

```
(0 IF SUB) OrElse T

((T TIMES 1000) IF 5 INLIST 2 4 6) OrElse ND

ROUND T DIVIDE 1000

norm T

( T IF T > 0 ) ELSE ( ND IF T < 0)
```

♢ If you have a secondary cube `xT` of the same shape as the current one, you may reference both, as in this example:

```
100 TIMES ( T MINUS xT ) DIVIDE xT
```

Note, however, that you are always assigning the result to the *current* cube.

User options

- A formula that assigns a cube of the size of the current cube (or else a single value).

Errors and warnings

- Error - Shape mismatch - Global computation fails.
- Error - Mismatched parentheses or brackets.

Related topics

Topic	Section	Page
COMPUTE	3.3.1	35
BROWSE	3.2.1	26

3.3.5 NORM

Function

Make a reference element equal to 100 and re-scale the others. Obtain index numbers.

Motivation

This command will normalize all values in the current cube, expressing them as a percentage of some reference element (which is set to 100).

The module lets you choose the dimension and the element to be set to 100, for example one of these:

Total revenues	= 100
Year 2005	= 100
All companies	= 100

The cube acquires a new dimension, with two elements `Absolute` and `Norm on` *Label*.

User options

- Dimension (where the reference element resides)
- Reference element

Side effects

- The current cube acquires an extra dimension.

Related topics

Topic	Section	Page
`norm` keyword	3.3.1.1	39
COMPUTE	3.3.1	35
GLOBAL	3.3.4	47

3.3.6 CONTRIBUTE

Function

3-way Variance breakdown.

Compute a formula for two time periods and analyze the contribution of each variable in the formula to the final result. Analyze the sensitivity of a result to its inputs.

Motivation

Suppose you are comparing two scenarios, say Budget and Actual, and find that Actual is $ 11900 above Budget.

The variable you are analyzing is a Margin, which is defined with the **COMPUTE** formula

```
("Quantity" TIMES "Unit Price")  MINUS "Cost of Goods"
```

The next question comes rather natural: what part of the $ 11900 increase (Actual vs Budget) is due to a higher Quantity, what to a higher Unit Price and what to a lower Cost of Goods ?

Since the formula at hand is not simply a sum (but, instead, contains a multiplication) there will also be a combined effect of Quantity and Unit Price.

This is a kind of routine analysis when comparing two scenarios, which can be Actual and Budget, as in the example above, or January and February, or any other pair you may need to compare.

♢ The way TANGRAM supports you in these exercises is the following.

It is assumed that you are computing a new variable (like Margin) on dimension 1 and that your scenarios (like Budget and Actual) are on dimension 2.

This is no serious constraint, because you can pivot your cube at any time with the **TRANSPOSE** command.

You can have as many elements as you like on the three dimensions: that is, on dimension 2 you must have two scenarios to compare and you may have extra elements that are unaffected by the **CONTRIBUTE** operation.

The execution of **CONTRIBUTE** will apply a formula and create an extra element both on dimension 1 and 2.

The new variable will accommodate the result of the formula and the new scenario will contain the deltas, that is, the contribution of each input variable to the result.

Dimension 3 is unaffected by **CONTRIBUTE**: all its elements are processed in a single execution.

Contributions are meant to be *absolute* contributions (like the $ 11900 in the example), but it is a simple matter to obtain *percent* contributions instead (via **COMPUTE** or **NORM**).

Example 1

Current cube: 5 variables, 3 scenarios, 1 product
Contents:

Prod.XYZ	Actual	Budget	Forecast
Quantity	1300	1100	990
Unit Price	40	38	38
Cost of Goods	17000	18500	18000
Commission	40	40	40
Delivery	35	20	20

After the **CONTRIBUTE** command, used with the formula

```
("Quantity" TIMES "Unit Price") MINUS "Cost of Goods"
```

your current cube will look like this:

Prod.XYZ	Actual	Budget	Forecast	Contr. Budget/Actual
Quantity	1300	1100	990	7600
Unit Price	40	38	38	2200
Cost of Goods	17000	18500	18000	1500
Commission	40	40	40	0
Delivery	35	20	20	0
Margin	35000	23300	19620	400

Note that the contributions of Commission and Delivery are nil, since these variables are not referenced in the formula.

Note also that the bottom-right cell of the result (400 in this example) represents the combined effects of an increase in Quantity, Unit Price and Cost of Goods.

As a concluding remark, note also that the Quantity row is likely to be expressed in items or boxes. The new column "Contr. Budget/Actual", however, contains delta Margins, so that it is expressed in Margin units ($ for example).

♢ While the initial motivation for this command was found in business topics similar to Example 1, the command **CONTRIBUTE** has been cast in a very general form and lends itself to numerous other applications.

For example, if the two scenarios have the meaning of a reference situation and the same situation with a unit increase then what you obtain in the third scenario are the deltas due to a unit increase.

Prod.XYZ	Actual	Actual+1	Sensitivity
Quantity	1300	1301	41
Unit Price	40	41	1301
Cost of Goods	17000	17001	-1
Commission	40	41	0
Delivery	35	36	0
Margin	35000	36340	-1

According to your trade, you may call this operation *sensitivity analysis* or *partial differenciation* or *variance breakdown* or *finite deltas*.

The formula that you use can have any complexity and follows the same rules as for the **COMPUTE** command.

The new label on dimension 2 (like, "Contr. Budget/Actual") is obtained from the labels of the two selected scenarios. Care must be taken to rename it appropriately, to ease the interpretation of the resulting printouts.

User options

- Label of the new element on dimension 1
- Formula on dimension 1
- Selection of two elements on dimension 2

Errors and warnings

- Error - An empty label is not allowed
- Error - Mismatched parentheses
- Error - Reference to non-existing elements
- Error - Illegal use of the double quote notation
- Error - Result has unacceptable size
- Error - Formula execution failed

Related topics

Topic	Section	Page
COMPUTE	3.3.1	35

3.4 Cube structure

3.4.1 TRANSPOSE

Function

Transpose the dimensions of the current cube.

Motivation

This module lets you transpose (or rotate, topple,...) the current cube, in the sense that you permute the order of its dimensions.

If you have ACCOUNTS, MONTHS, COMPANIES, you might prefer to view it as COMPANIES, ACCOUNTS, MONTHS.

This operation is unnecessary for the sole purpose of printing, because all printing modules feature a transposition on-the-fly.

The typical motivation for this command is to bring your cube to Standard Orientation, where the dimensions have the specialized meaning of Variables, Periods, Items.

You may need **TRANSPOSE** to bring a dimension to the conventional position (1st, 2nd, 3rd) so that dimension-specific modules (like **TREND**, or **QUERY**) produce what you expect.

User options

- New dimension order (a permutation)

Related topics

Topic	Section	Page
CROSSOVER	3.4.2	53
SHUFFLE	3.4.9	63

3.4.2 CROSSOVER

Function

Move a facet (*alias*: a code, a key) in the labels to a different dimension and restructure the current cube accordingly.

Motivation

As a rule, TANGRAM operations act on each dimension independently.

This is a fundamental simplification, because it lets you concentrate on a dimension at a time (see **COMPUTE**, for example).

The complexity that you handle is that of a *mono-dimensional* structure.

TANGRAM's unique **CROSSOVER** command handles more sophisticated cases in which this separation of dimensions becomes a problem.

CROSSOVER implements a very general transformation of cube structures: it lets you point to some label positions (a facet) and move them to a different dimension, taking care of the numeric cells that must be re-associated with the proper labels.

Example 1

At the start you have these labels on dimension 1:

```
costs     jan
revenues  jan
profit    jan
costs     feb
revenues  mar
```

And these labels on dimension 2:

```
France
Germany
Italy
```

You use **CROSSOVER** to move the month facet to dimension 2. The program will first generate all missing Variable—Month pairs, expanding the cube with missing values:

```
costs     jan
revenues  jan
profit    jan
costs     feb
revenues  feb
profit    feb
costs     mar
revenues  mar
profit    mar
```

Then it will move the month code to dimension 2, obtaining this final situation.

Dimension 1:

```
costs
revenues
profit
```

Dimension 2:

```
feb   France
feb   Germany
feb   Italy
jan   France
jan   Germany
jan   Italy
mar   France
mar   Germany
mar   Italy
```

♢ Note that the total number of cube cells is preserved (if all key pairs were already present) or increased.

After execution, the elements of the target dimension are sorted on the labels.

Example 2

It should be clear that **CROSSOVER** is specially useful when numerous facets have been accommodated onto TANGRAM's dimensions.

For example *accounts, months, branches, products, markets, channels* may be first structured (in a 3-way cube) as

```
ACCOUNTS/PRODUCTS
MONTHS/BRANCHES
CHANNELS/MARKETS
```

and you later need to restructure the cube as

```
ACCOUNTS/MONTHS
PRODUCTS/BRANCHES
CHANNELS/MARKETS
```

The **CROSSOVER** command guarantees that in these more complex transformations each cell is always associated to the appropriate labels.

The dimension names are not updated by **CROSSOVER**. You may want to create a physical database with **SAVEAS** and then update them by hand.

User options

- the dimension that must shrink (donates a facet)
- the dimension that must grow (receives a facet)
- the label positions that migrate

Errors and warnings

- Error - Duplicate labels on a dimension

Related topics

Topic	Section	Page
BUILDDB	4.3.4	100
CROSS	5.9	119
TRANSPOSE	3.4.1	52
SHUFFLE	3.4.9	63

3.4.3 DELTA

Function

Compute the absolute and percent difference between current and secondary cubes.

Motivation

This module can be handy to compute absolute and percent differences between two versions of the same situations, like Actual and Budget.

The module requires a current and secondary cube of the same rank and size and leaves its results in the current cube.

The module accepts two inputs, that is, the label of the current and secondary cubes (like “New” and “Old”).

The current cube acquires an extra dimension to accommodate both cubes and the absolute and percent difference. The new dimension has four elements, for example:

```
New
Old
New-Old
New-Old %
```

User options

- Current cube label
- Secondary cube label

Errors and warnings

- Error - Current and secondary cube must have the same rank and size.

Related topics

Topic	Section	Page
COMPUTE	3.3.1	35
CROSSOVER	3.4.2	53

3.4.4 BLOWUP

Function

3-way Blowup (collapsed dimensions of) current cube using weights in the secondary cube.

Motivation

When a dimension of the current cube reduces to a single element, you may want to spread or allocate the values to an element population, in proportion to elements existing in the secondary cube.

This can be done on 1, 2 or 3 dimensions in a single execution.

The rule is that, before the execution of **BLOWUP**, every dimension of the current cube must either be equal to that of the secondary cube or reduce to one element (the former is respected, the latter blown up).

The result consists in a cube of the same level of detail as the secondary cube, in which labels of exploded dimensions are borrowed from the secondary cube.

Example 1

Attribute the seasonal behavior of sales (secondary cube) to training costs (current cube) that are known only for the whole year.

Current cube: 120 accounts, 1 period, 40 branches

Secondary cube: 120 accounts, 12 periods, 40 branches

After the execution of **BLOWUP**:

Current cube: 120 accounts, 12 periods, 40 branches.

You can check the correctness of the operation in two ways:

- summing all periods in the current cube, you re-obtain the previous current cube.

- selecting a single account and a single branch you obtain 12 values that are proportional to corresponding weights in the secondary cube.

Example 2

We wish to obtain a Profit and Loss for each customer-product pair; revenues are available at this level of detail but costs are available only at the product level.

One can *spread* the costs over the various customers assuming that they are proportional to some variable that is available in detail (for example Revenues from Sales).

Example 3

The current cube may contain a single value (a $1 \times 1 \times 1$ cube) that represents a forecast of next year's result.

To blow it up to full size, one may use this year's actual values as an expedient and reasonable assumption for all the missing details.

Errors and warnings

- Error - There is no secondary cube
- Error - Element count mismatch between current and secondary cube
- Warning - Weights in the secondary cube add up to zero

Related topics

Topic	Section	Page
DISAGGREGATE	3.4.5	57
ROLLUP	3.3.2	43

Note. **DISAGGREGATE** is a generalization of **BLOWUP** in the case of more complex relationships between the labels.

3.4.5 DISAGGREGATE

Function

3-way Disaggregate the current cube using weights in the secondary cube.

Motivation

This module produces more detail than you have, under the assumption that the allocation is done in proportion to some other cube (*a driver*).

The current cube is the candidate for disaggregation, the secondary cube is used as a template.

The detailed relationship between the elements is controlled by some matching codes in their labels.

The concept is similar to that of the **BLOWUP** command, with more freedom to define the association between elements.

The disaggregation is done on all dimensions in a single execution.

Example 1

You know the *sales revenues* at the *province* level and the *costs* at the *country* level.

Suppose, further, that you want to estimate (or spread) the costs at the province level, assuming that they are proportional to sales revenues.

This is easily achieved, if you place your sales revenues in the secondary cube and make sure that province names are accompanied by their country code (to define a many-to-one relationship).

The current cube will hold cost data (with country codes).

The **DISAGGREGATE** command will ask you to point to the positions of country codes in both cubes (for instance, on the third dimension of both cubes).

A similar many-to-one relationship may exist on the first and second dimension (or, in a simple case, these dimensions may be identical in both cubes).

After execution of the command you will find your costs split by province.

To help you keep track of what happened, the new labels will contain *both* old and new labels. For example:

```
MI Milano|IT Italy
```

Should you apply the **ROLLUP** command after **DISAGGREGATE**, to roll up again to the country level, you would restore the original cube.

♢ The **DISAGGREGATE** command is carefully designed to handle all extreme cases. One of these is when the codes that define the many-to-one relationship have zero length.

This is accepted, and amounts to defining a one-to-all relationship.

Example 2

The first dimension of the current cube contains a single element:

```
Total overhead
```

whereas the first dimension of the secondary cube contains:

```
Personnel costs
Office costs
Operational costs
```

then a zero-length code means that the three costs (secondary cube) are details of the `Total overhead` (current cube) and the `Total overhead` must be split into three component elements.

User options

- Code positions within the labels of each dimension of both cubes (six codes in all).

Side effects

- Implicit input: current and secondary cube
- Implicit output: current cube
- The labels of the current cube are updated.

Errors and warnings

- Error - Code length mismatch on corresponding dimensions.
- Error - Some labels of the current cube do not find a counterpart in the secondary cube
- Warning - Some labels of the secondary cube are not referenced.
- Warning - The secondary cube contains negative values.

Note. If the cells of the secondary cube corresponding to a single code add up to zero (or are all missing), no explicit warning is issued but the corresponding elements of the result will be missing.

Related topics

Topic	Section	Page
ROLLUP	3.3.2	43
BLOWUP	3.4.4	56
`norm` keyword	3.3.1.1	39

3.4.6 MERGE

Function

Merge current and secondary cube, on the basis of equal labels.

Motivation

The function of the **MERGE** command is to merge the cells of the current and secondary cube, assuming that equal labels imply equal meaning.

The command requires two cubes of the same rank and updates both to align their labels on all dimensions.

In particular the labels of each dimensions are the *set union* of the original labels.

In other words, the labels of the result are the collection of labels from the secondary cube (in the same order), followed by the labels found only in the current cube (in the same order).

The program will report on the merging of the two sets of labels (whether they were disjoint, intersecting or coincident).

If both input cubes contain a cell associated with the same labels on all dimensions, we may have three cases:

- The cell has a missing value in at least one cube.
- The cell contains the same value in both cubes.
- The cell contains different values (comparison tolerance is 0.1).

Only the third case is regarded as a conflict. After a warning, the value in the secondary cube prevails.

Note that no information is lost, because the current cube still contains the conflicting cells.

The result will have missing values where both input cubes have missing values.

Example 1

Current cube: 5 accounts, 6 months, 1 company
Contents:

XYZ ltd	jan	feb	mar	apr	may	jun
income	300	200	500		350	
var. cost	120	170	220	170		170
fixed cost	40	40	40	40	40	40
margin	35	30				
profit	10	12				

Secondary cube: 4 accounts, 3 months, 1 company
Contents:

XYZ ltd	dec	jan	feb
var. cost	95	120	170
profit		11	12
margin	28		30
personnel	17	16	17

After the **MERGE** command, you find the secondary cube to be:

XYZ ltd	dec	jan	feb	mar	apr	may	jun
var. cost	95	120	170	220	170		170
profit		**11**	12				
margin	28	35	30				
personnel	17	16	17				
income		300	200	500		350	
fixed cost		40	40	40	40	40	40

The program will also warn you that a single cell was in conflict in the two cubes (see profit of january).

Example 2

You constitute two TANGRAM cubes with 2006 and 2007 data respectively. The collections of Customers and Products in the two years are unrelated but use the same codes.

You then want to study the evolution of those customers and products that existed in both years.

The **MERGE** command is the main tool for this analysis, because it will return a two-period data cube and take care of all code-matching.

Example 3

Note that merging timeseries with different periodicity is just a special case of **MERGE**.

You only have to use a consistent period labelling, for, say, end-of-month, end-of-quarter and end-of-year values.

For example, if the time periods in the current cube are

```
Jan 2007
Feb 2007
Mar 2007
Apr 2007
May 2007
Jun 2007
```

and the secondary cube contains quarterly data

```
Mar 2007
Jun 2007
```

then the two periodicities are automatically merged.

Side effects

- Implicit input: the current and secondary cube
- Implicit output: the current and secondary cube with equal labels. The secondary cube represents the merged output.

Errors and warnings

- Error - There is no secondary cube
- Error - Duplicate labels in the current cube
- Error - Duplicate labels in the secondary cube
- Warning - Redundant values in *n* cells

- Warning - Conflicting values in *m* cells

Related topics

Topic	Section	Page
TRANSFER	4.3.3	97
`COMMON` keyword	4.3.3	97
`OrElse` keyword	3.3.1.1	39

3.4.7 PROMOTE

Function

Add an extra dimension to the current cube. The extra dimension is the last and entails a single element.

User options

- Dimension name
- Element label

Errors and warnings

- Error - 15 dimensions is a system maximum.

Related topics

Topic	Section	Page
DEMOTE	3.4.8	62
CROSSOVER	3.4.2	53
SHUFFLE	3.4.9	63

3.4.8 DEMOTE

Function

Remove the last dimension of the current cube, provided it consists of a single element.

Errors and warnings

- Error - 3 dimensions is a system minimum.
- Error - The last dimension must reduce to a single element.

Related topics

Topic	Section	Page
PROMOTE	3.4.7	62
CROSSOVER	3.4.2	53
SHUFFLE	3.4.9	63

3.4.9 SHUFFLE

Function

Restructure the cube to a new rank, creating, transposing, joining dimensions.

Motivation

This module defines the new destination of existing dimensions in a new cube structure.

The same result of **SHUFFLE** could be achieved by a long sequence of **PROMOTE, CROSSOVER, TRANSPOSE** ...

Typical application: collapse an N-way cube to a 3-way cube to use modules that only work 3-way.

Example

Start with a 4-way current cube, with dimension names

```
[1] Measures
[2] Periods
[3] Products
[4] Versions
```

If you code the destinations as `1 2 6 1` you obtain a new cube of rank 6, where dim 1 is the cross of former dims 1 and 4, dim 2 is unaffected, dims 3 4 5 are empty, dim 6 is the former dim 3.

The new dimension names will be:

```
[1] Measures Versions
[2] Periods
[3]
[4]
[5]
[6] Products
```

Since dimensions with no name can be confusing, you probably want to save this cube (**SAVEAS**) and name the empty dimensions (**UPDDBLABELS**).

User options

- New destinations of the current dimensions (a vector of positive integers, as many as the rank of the current cube).

Errors and warnings

- Error - Operation impossible - Duplicate elements on dimension *dim*
- Error - Too many dimensions - Maximum 15

  (The sum of the cube ranks before and after the operation must not exceed 15)

- Error - 3 dimensions is a system limit
- Error - Illegal target dimensions - Expects *N* integers

Related topics

Topic	Section	Page
PROMOTE	3.4.7	62
DEMOTE	3.4.8	62
TRANSPOSE	3.4.1	52
CROSSOVER	3.4.2	53

3.5 Cube output to file

3.5.1 SAVEAS

Function

Create a new physical database that stores the current cube.

Motivation

This command transforms the current cube into a new TANGRAM database.

Its typical use is to permanently store the result of processing or else to save intermediate cubes on which you can fall back in case of a blunder.

The new database will have current size equal to maximum size.

The **SAVEAS** command does not make the new database current.

User options

- Name of the new database

Related topics

Topic	Section	Page
NEWDB	4.2.2	91
BUILDDB	4.3.4	100
CROSSDB	4.2.3	92
DOCK	4.3.7	105
STORE	3.8.1	85

3.5.2 UNLOAD

Function

3-way Unload the current cube as a flat file.

Motivation

This module lets you produce a four-field flat file that contains the labels and cells of the current cube.

Three fields in each record will contain (part of) the labels of a cell, while the forth will contain the numeric value.

Missing values are not unloaded. The number of records in the file will thus equal the number of non-missing cells in the current cube.

Numerical values that exceed the declared width are represented as a sequence of asterisks.

User options

- Key positions on dim. 1
- Key positions on dim. 2
- Key positions on dim. 3
- Width of numeric field
- Number of decimal places
- Name of the flat file `.txt`

Errors and warnings

- Warning - Numeric field too narrow in n cases.
- Warning - n missing values did not produce a record.

Related topics

Topic	Section	Page
LOADDB	4.3.2	96
EXPORT	3.5.3	66
MASKPRINT	3.6.2	74

3.5.3 EXPORT

Function

3-way Export the current cube in a comma-separated format.

Motivation

This command will let you export the current cube to a flat file that acts as an exchange format to other software, like a spreadsheet.

The command will export both cells and labels of the current cube in a two-dimensional format (see examples below).

Labels and cells are separated by user-chosen characters. If these are chosen to reflect the current choice in your spreadsheet, then the file can immediately be loaded by the spreadsheet and correctly mapped into cells.

Example

The following are two examples of flat files obtained from the same cube with different views. The separator characters are `'[]|,'`

```
[OPTILUX SpA    ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[TOT. REVENUES  ]  |10086,00|21490,00|33697,00|45189,00
[FIRST MARGIN   ]  | 3061,00| 6815,00|11045,00|14750,00
[SECOND MARGIN  ]  | -215,17|  152,75|  834,39|10100,25
[RESULT         ]  | -503,62| -388,11|  -41,28| 8898,35
[PROFIT BEF.TAX ]  |-1106,29|-1528,20|-1960,88| 6183,77
[PROFIT AFT.TAX ]  |-1095,99|-1507,59|-1960,88| 6178,62
[SUNNY Sarl     ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[TOT. REVENUES  ]  |    0,00| 2622,00| 4778,00| 7232,00
[FIRST MARGIN   ]  |    0,00|  322,00|  810,00| 1420,00
[SECOND MARGIN  ]  |    0,00|-1082,87|-1522,16|-1869,84
[RESULT         ]  |    0,00|-1184,18|-1666,39|-2091,33
[PROFIT BEF.TAX ]  |    0,00|-1125,80|-1611,44|-2046,69
[PROFIT AFT.TAX ]  |    0,00|-2259,02|-3235,73|-4110,53
[HITEL Ltd      ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[TOT. REVENUES  ]  |    0,00|17268,00|25912,00|33516,00
[FIRST MARGIN   ]  |    0,00| 4444,00| 6471,00| 8180,00
[SECOND MARGIN  ]  |    0,00|  865,03| 1008,50|  893,44
[RESULT         ]  |    0,00|  702,27|  745,11|  506,93
[PROFIT BEF.TAX ]  |    0,00|  631,76|  650,19|  394,87
[PROFIT AFT.TAX ]  |    0,00|  907,73|  655,15|  683,03
[TAIPEI Spa     ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[TOT. REVENUES  ]  |    0,00|    0,00|86044,00|********
[FIRST MARGIN   ]  |    0,00|    0,00|27838,00|37791,00
[SECOND MARGIN  ]  |    0,00|    0,00| 8113,27|11673,27
[RESULT         ]  |    0,00|    0,00| 5080,65| 6600,99
[PROFIT BEF.TAX ]  |    0,00|    0,00| 4111,70| 6074,21
[PROFIT AFT.TAX ]  |    0,00|    0,00| 5214,06| 8056,31
[MICRO SA       ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[TOT. REVENUES  ]  |    0,00|    0,00|81705,00|********
[FIRST MARGIN   ]  |    0,00|    0,00|39593,00|54675,00
[SECOND MARGIN  ]  |    0,00|    0,00| 1475,45| 2088,53
[RESULT         ]  |    0,00|    0,00|-3450,62|-4248,91
[PROFIT BEF.TAX ]  |    0,00|    0,00|-3831,80|-4664,43
[PROFIT AFT.TAX ]  |    0,00|    0,00|-5713,63|-7356,68
```

```
[TOT. REVENUES  ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[OPTILUX SpA    ]  |   10086|   21490|   33697|   45189
[SUNNY Sarl     ]  |       0|    2622|    4778|    7232
[HITEL Ltd      ]  |       0|   17268|   25912|   33516
[TAIPEI Spa     ]  |       0|       0|   86044|  118146
[MICRO SA       ]  |       0|       0|   81705|  111790
[FIRST MARGIN   ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[OPTILUX SpA    ]  |    3061|    6815|   11045|   14750
[SUNNY Sarl     ]  |       0|     322|     810|    1420
[HITEL Ltd      ]  |       0|    4444|    6471|    8180
[TAIPEI Spa     ]  |       0|       0|   27838|   37791
[MICRO SA       ]  |       0|       0|   39593|   54675
[SECOND MARGIN  ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[OPTILUX SpA    ]  |    -215|     153|     834|   10100
[SUNNY Sarl     ]  |       0|   -1083|   -1522|   -1870
[HITEL Ltd      ]  |       0|     865|    1008|     893
[TAIPEI Spa     ]  |       0|       0|    8113|   11673
[MICRO SA       ]  |       0|       0|    1475|    2089
[RESULT         ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[OPTILUX SpA    ]  |    -504|    -388|     -41|    8898
[SUNNY Sarl     ]  |       0|   -1184|   -1666|   -2091
[HITEL Ltd      ]  |       0|     702|     745|     507
[TAIPEI Spa     ]  |       0|       0|    5081|    6601
[MICRO SA       ]  |       0|       0|   -3451|   -4249
[PROFIT BEF.TAX ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[OPTILUX SpA    ]  |   -1106|   -1528|   -1961|    6184
[SUNNY Sarl     ]  |       0|   -1126|   -1611|   -2047
[HITEL Ltd      ]  |       0|     632|     650|     395
[TAIPEI Spa     ]  |       0|       0|    4112|    6074
[MICRO SA       ]  |       0|       0|   -3832|   -4664
[PROFIT AFT.TAX ]  |[Jan 07]|[Feb 07]|[Mar 07]|[Apr 07]
[OPTILUX SpA    ]  |   -1096|   -1508|   -1961|    6179
[SUNNY Sarl     ]  |       0|   -2259|   -3236|   -4111
[HITEL Ltd      ]  |       0|     908|     655|     683
[TAIPEI Spa     ]  |       0|       0|    5214|    8056
[MICRO SA       ]  |       0|       0|   -5714|   -7357
```

User options

- the mapping of TANGRAM's three dimensions to lines, columns, pages
- the field width of the numeric format
- the number of decimal places

- the separator characters (start of label, end of label, cell separator, decimal point).
- the name of the flat file `.txt`

Errors and warnings

- Warning - n missing values have been exported as zeroes

Related topics

Topic	Section	Page
UNLOAD	3.5.2	65
BROWSE	3.2.1	26
MASKPRINT	3.6.2	74

3.5.4 TOHTM

Function

3-way HTML by example - Unload the cube as an HTML document according to a template.

Motivation

This module exports the current cube in the form of HTML code (*alias*: WWW or WEB pages) that can be viewed with a browser like Netscape or Internet Explorer.

This means you can produce hypertext presentations of TANGRAM data with negligible effort.

A single cube produces a single HTML file where two dimensions become rows and columns and the third becomes an intra-document index.

Note that some cube transpositions are more appropriate than others and the total size of the `.htm` file should not exceed some 500 kbytes for fast loading.

Since huge tables are difficult for the browser to handle, TANGRAM lets you specify the maximum table size.

A template is a HTML file that defines your formatting preferences. In particular the template must contain a one-by-one table that specifies the desired styling.

User options

- Name of the template file that defines the HTML style, with extension `.tmp`
- Maximum row and column count (in each HTML TABLE construct)
- Decimal places

- Document title (replaces the symbolic variable `@title` that may appear in the template)
- Mapping of TANGRAM's three dimensions to lines, columns, pages
- Name of output file, with extension `.htm`

Errors and warnings

- Error - The template does not contain eight blank (separation) lines
- Error - The analysis of the template file was unsuccessful

Related topics

Topic	Section	Page
TOLATEX	3.5.5	71
REPORT	3.6.1	73

3.5.4.1 Template design

Advanced You can define your own templates, keeping in mind that they must have a given format for TANGRAM to analyze them correctly.

In particular a template must have:

- An initial block of HTML code, with no blank lines
- Seven lines of code, separated by eight blank lines
- A final block of HTML code with no blank lines.

The file `map.tmp` (listed here), marks with an asterisk (`'*'`) the compulsory blank positions. Code segments between two asterisks, like `<IMG SRC="cw.gif">`, can be replaced by your choice.

```
<HTML> <HEAD> <TITLE>
@title
</TITLE>
</HEAD> <BODY BGCOLOR=#C0C0C0  TEXT=#000000 >
<CENTER> <H1>
@title
</H1> </CENTER>
<H2> WEB output example </H2><P>
This page was generated by TANGRAM, a DSS-OLAP system developed by
Codework Italia <P>
<A HREF="#TOC"> <IMG ALIGN=center SRC="toc.gif"> </A>
N.B. This button brings you back to the document's Table of
Contents, that is, to this point.
```

```
You will find one after every table.
<HR>
**************************************************************
<H2><A NAME="TOC">*Contents*</A></H2>
**************************************************************
<CENTER><H2><A NAME="1"> Page-title </A></H2></CENTER>
**************************************************************
<CENTER> <TABLE BORDER=3 CELLPADDING=5 CELLSPACING=5>
**************************************************************
<CAPTION ALIGN=bottom><B>*Page-caption*</B></CAPTION>
**************************************************************
<TR>*<TH >*<IMG SRC="cw.gif">*</TH>*<TH>*col-header*</TH>*</TR>
**************************************************************
<TR>*<TH >*line-header       *</TH>*<TD>*number    *</TD>*</TR>
**************************************************************
</TABLE> </CENTER>
**************************************************************
<P><HR SIZE=3 WIDTH=85%> <P>
<CENTER> <TABLE BORDER=3 CELLPADDING=5 CELLSPACING=5>
<CAPTION ALIGN=bottom><B> TANGRAM </B></CAPTION>
<TR> <TH> <A HREF="http://www.codework.it/tangram">Homepage</A></TH>
<TH> <A HREF="cdwk.htm"> Top document </A> </TH>
<TH> <A HREF="mailto:support@codework.it">Feedback</A> </TH></TR>
</TABLE> </CENTER>
<P><HR SIZE=3 WIDTH=85%> <P>
</BODY></HTML>
```

Remember that the initial and final block can be anything (that a browser accepts), while the seven central lines must have the right count of HTML segments.

The variable `@title` , if present, must be followed by 44 blanks to make room for a 50 char. title.

Note that several HTML files generated by TANGRAM can be assembled by a main HTML page that contains links to all of them.

Better still, a navigation structure can be defined in the bottom block of the template, so that the work of the hypertext author is simplified and the reader finds consistent navigation aids in all documents.

3.5.5 TOLATEX

Function

3-way Unload the current cube as a **LaTeX** source document.

Motivation

LaTeX is a well known document preparation system which in turn is based on TeX. Like HTML and XML, it processes text interspersed with structure and stlyling tags.

The manual you are reading is an example of a LaTeX document.

This module produces a LaTeX source file that can be automatically compiled and printed (or transformed into **PostScript** or **Acrobat** format).

You don't need any LaTeX expertise to take advantage of TANGRAM's output but you still must have LaTeX installed somewhere.

Since very large tables are difficult to fit into printed pages, the module will let you choose the maximum number of rows and columns of each LaTeX table and iterate on the TANGRAM cube accordingly.

User options

- Maximum number of rows and columns per LaTeX table
- Number of decimal places
- Main document title
- Second document title
- Third document title
- Mapping of TANGRAM's three dimensions to lines, columns, pages
- A list of rows that must be highlighted
- Landscape orientation of the printer (as opposed to Portrait)
- Output file name `.tex`

Errors and warnings

- Warning - Text contains TeX reserved characters :

Related topics

Topic	Section	Page
TOHTM	3.5.4	69

3.6 Printed outputs

Overview

There are many ways to print TANGRAM cubes.

The most popular is to access TANGRAM databases from Excel (via the TANGRAM-reader) and map TANGRAM data on *styling templates*, that is, Excel sheets that contain only titles and cell styling options, such as color, font, frames,...

This section presents the more traditional ways to produce fixed-font boxed printouts.

The advantages of this choice are mainly speed and versatility, as it is easier to find solutions that do not require any maintenance when the size and orientation of the cube vary.

If you send the output of these modules to a file, it will be appended to file `log\sysprint.log`, that you can manage (empty, copy, delete, print) with non TANGRAM services.

Remember that, containing framing characters, this file is coded in ASCII (as opposed to ANSI) and it is best viewed with the internal TANGRAM editor (Main menu) or with ASCII editors (like DOS Edit) rather than ANSI editors (like Notepad, Wordpad, Word).

If you send the output to the standard printer, the orientation (Landscape or Portrait) is controlled by Windows, not by TANGRAM. The number of lines per page (and hence the character size) is chosen by TANGRAM (See Main menu, Configuration, Miscellany, Printed page).

The keyword `DEFAULT` returns the maximum character count per page, (rows and columns) which depends on the current printer settings. As such, it can be used in replies to TANGRAM.

For ex.

```
   DEFAULT
80 93
```

3.6.1 REPORT

Function

3-way Print the current cube in a page-oriented format.

Motivation

This is one of several ways to produce printed output.

REPORT produces the traditional boxed presentations, where row, column and page headings are derived from the cube labels.

This module is a good candidate for producing printouts when:

- the production must be automated, irrespective of the cube size
- options and customization can be reduced to a minumum
- page design is stable.

The main idea is to specify the page height and width (in characters) and let the program iterate on all dimensions of the current cube to produce as many printed pages as needed.

The header and footer text of each page may contain the keywords
`@date` `@time` `@page`
which will be replaced by their current values.

User options

- Mapping of TANGRAM's three dimensions to lines, columns, pages
- Field width (it may be a string of values, to format columns differently)
- Decimal places (it may be a string of values, to format columns differently)
- List of underlined rows (indices on a cube dimension that is printed on lines)
- Page size in characters (height and width). Reply with the keyword `DEFAULT` if you just want to fill all available space on the physical page.
- Header text (one line)
- Footer text (one line)
- Destination (to screen, printer, file). Three Boolean values, for ex. `ON,ON,OFF`

Related topics

Topic	Section	Page
TOHTM	3.5.4	69
BROWSE	3.2.1	26
MASKPRINT	3.6.2	74
DRAFTPRINT	3.6.3	77

3.6.2 MASKPRINT

Function

3-way Format by example - Print the current cube according to a detailed layout taken from a mask file.

Motivation

This is one of several ways to produce printed output.

MASKPRINT merges a mask file (that describes the page in full detail on a char-by-char basis) with the contents of the current cube.

This module is a good candidate for producing printouts when

- a fixed font output is acceptable
- the production must be automated
- the page layout must be defined in full detail, with framed and boxed areas and fixed text
- the page design and the current cube size are stable.

The basic idea is to prepare a page layout with an ASCII editor, marking the position of labels and numeric fields with reserved characters.

This means that an end user may maintain his masks with a simple editor and no programming is involved.

The program will take care of the complex mapping between the current cube and the mask, iterating the operation on all dimensions until the cube contents have been accommodated on printed pages.

The mask file `report.msk` is a reserved name, in the sense that every execution of **REPORT** rewrites it with a mask corresponding to its current page layout. You can thus use **REPORT** to obtain a first approximation to your page layout and then refine it by hand before using **MASKPRINT**.

The mask file may contain the keywords `@date` `@time` and `@page` which will be replaced by their current values.

♢ MASKPRINT is not simply a way to obtain extravagant framed pages. The mask may also contain the source code required by some software product and the reserved characters mark positions where TANGRAM data must be placed. Or, the mask may be a record layout for exporting the cube.

User options

- Mapping of TANGRAM's three dimensions to lines, columns, pages
- Name of the mask file (extension `.msk`)
- Destination (to screen, printer, file)

Errors and warnings

- Warning - The mask does not contain row descriptors
- Warning - The mask does not contain column descriptors
- Warning - The mask does not contain page descriptors
- Warning - Multiple page descriptors
- Warning - Row descriptors are not aligned with numeric fields
- Warning - Column descriptors are not aligned with num. fields
- Warning - Numeric fields are not aligned in an array
- Warning - The mask does not contain numeric fields!
- Warning - The numeric field count varies from line to line!
- Error - Invalid mask format

Related topics

Topic	Section	Page
TOHTM	3.5.4	69
BROWSE	3.2.1	26
REPORT	3.6.1	73
DRAFTPRINT	3.6.3	77
EXPORT	3.5.3	66

3.6.2.1 Mask design

Advanced You can design your masks (*alias*: forms, templates) with any ASCII editor of your choice or with TANGRAM (Main menu).

You do so by entering fixed text, frames, boxes embedded with *reserved* characters that mark the positions that must accommodate labels and values.

The template file must have the extension `.msk` and may contain the following reserved characters:

$	position of row labels
&	position of column labels
?	position of page labels
!	in column 1 marks a comment line
#	position of a numeric field
.	decimal dot within a numeric field

Example

```
        Revenues from sales

        Branch:    ??????????????
        -------------------------
                                          <----forecast---->
                  &&&&&&&&   &&&&&&&&   &&&&&&&&   &&&&&&&&

  $$$$$$$$$$$   #####.##   #####.##    #####.#    #####.#
  $$$$$$$$$$$   #####.##   #####.##    #####.#    #####.#
  $$$$$$$$$$$   #####.##   #####.##    #####.#    #####.#
  $$$$$$$$$$$   #####.##   #####.##    #####.#    #####.#

  N.B. all values in thousand dollars

! Form   AZ04   B.M. created 4/5/2007
! Reporting procedure XYZ
```

You can specify a different number of decimal places in each field.

The width of the cube labels needs not match the receiving field in the mask (labels are truncated or padded as needed).

The labels associated with the cells can be taken from the current cube or can be "frozen" in the template file.

You are expected to create masks that help interpreting the values with suitable row and column headings.

A warning is issued when this principle is not respected and reserved characters are irregularly laid out. If your mask does not contain an orderly array of numeric fields (with the same field count on each line) then the mask is rejected.

Error example

```
      Branch:    ?????????????
      -------------------------
                                    <----forecast---->
             &&&&&&&&   &&&&&&&&  &&&&&&&&  &&&&&&&&

$$$$$$$$$$   #####.#    #####.#   #####.#   #####.#
             #####.#                        #####.#
$$$$$$$$$$   #####.#    #####.#   #####.#   #####.#
             #####.#    #####.#   #####.#
$$$$$$$$$$                          ###########
```

3.6.3 DRAFTPRINT

Function

3-way Print the current cube in a draft printout.

Motivation

This module will print the whole current cube over a logical page with fixed width and infinite height.

Column labels are printed to their full length, but they are slated to produce the most compact representation.

This module can also independently sort both rows and columns of each logical page according to the cube values.

The label of the element used for sorting are marked with ***.

If you do not want rows and columns to vary from page to page (that is, you do not want the sorting) reply `NONE` in the 'Sorting row' and 'Sorting column' field.

Example

Let your cube contain sales data, split by year, product and customer. You want a page for each year, the products down and the customers across the page.

On each page you want to sort the products on a column called 'All customers' and to sort the customers on a row called 'All products'.

The sorting order will, in general, depend on the year (and this rules out the possibility of sorting the current cube before printing).

User options

- Mapping of TANGRAM's three dimensions to lines, columns, pages
- Field width (one value)
- Decimal places (one value)
- Page width in characters
- Sorting row (a single element index or the keyword `NONE`)
- Sorting column (a single element index or the keyword `NONE`)
- Destination (to screen, printer, file).

Related topics

Topic	Section	Page
REPORT	3.6.1	73
MASKPRINT	3.6.2	74
`BYVALUE` keyword	6.2	127

3.7 Time-related operations

3-way This section groups all services that are time-specific, in the sense that they are meaningful only on a dimension that represents Periods.

If you move along the Periods dimension and keep the other coordinates fixed, you obtain a string of values that can be regarded as a *timeseries*.

This section, in other words, deals with timeseries operations.

All modules in this section apply to 3-way cubes and assume that your Periods are on dimension 2.

If this is not the case, use **TRANSPOSE** to rotate the cube as needed.

The dimension names (like 'Years' or 'Periods' or 'Quarters') are irrelevant.

3.7.1 GROWTH

Function

Compute the percent growth from a period to the next.

Motivation

This command computes the percent growth of every timeseries from a period to the next.

In mathematical notation, if we call X_n the contents of a cell in period n, the percent growth is defined as

$$100\frac{X_n - X_{n-1}}{\mid X_{n-1} \mid}$$

where X_{n-1} is the previous value.

The current cube acquires an extra dimension, with elements `Absolute` and `Growth from prev`

Missing values in the result may be due to missing values in the input cube or to zeroes in the previous period.

Side effects

- The current cube acquires an extra dimension.

Related topics

Topic	Section	Page
TREND	3.7.4	81
COMPUTE	3.3.1	35
FORECAST	3.7.5	82
keyword `PRE`	3.3.1.1	39
keyword `POST`	3.3.1.1	39

3.7.2 TODATE

Function

Compute To-Date values from Period-only values. Discrete integral.

Motivation

This module will transform *period* values into *to-date* values (*alias*: running totals, discrete integrals).

If, for example, a timeseries displays the following values in the first six months

```
3   4   3   3   5   4
```

then the to-date values will be:

```
3   7  10  13  18  22
```

The current cube acquires an extra dimension, with elements `Period` and `To date`.

A missing value in a timeseries produces missing values from that cell to the end of the timeseries.

All existing periods are treated in a uniform way and year boundaries are ignored: if your cube contains 24 months, the running totals will span two years.

If this is not what you intended, it is your responsibility to select appropriate periods before calling the module **TODATE**.

Side effects

- The current cube acquires an extra dimension.

Related topics

Topic	Section	Page
TOPERIOD	3.7.3	80
`todate` keyword	3.3.1.1	39
`toperiod` keyword	3.3.1.1	39

3.7.3 TOPERIOD

Function

Compute Period-only values from Todate values. Discrete differentiation.

Motivation

In numerous situations, you need two views of the same timeseries: absolute period values and "to-date" values, typically from the beginning of the year.

The **TOPERIOD** command computes period values, assuming that existing values are "to-date" (*alias*: running totals).

If, for example, a timeseries displays the following to-date values in the first six months

```
3  10  21  20  33  50
```

then the period values will be:

```
3   7  11  -1  13  17
```

The current cube acquires an extra dimension, with elements `To date` and `Period`.

A missing value in a timeseries produces missing values from that cell to the end of the timeseries.

The command accepts no options: it is the user responsibility to make sure that the operation is meaningful for all variables involved.

Note also that the operation applies to all periods in the current cube.

In most cases, one would select one year at a time for this kind of operation.

In other cases (like a project that spans several years) "to-date" values and differences over a longer time-span are meaningful.

Side effects

- The current cube acquires an extra dimension.

Related topics

Topic	Section	Page
TODATE	3.7.2	79
`toperiod` keyword	3.3.1.1	39
`todate` keyword	3.3.1.1	39

3.7.4 TREND

Function

Compute the trend from a raw timeseries.

Motivation

This module computes the *trend* of all timeseries in the current cube, to highlight significant long-term phenomena and discard noise or incidental fluctuations.

In mathematical notation, given a timeseries X_n, we define the trend as a second timeseries Y_n obtained as

$$Y_1 = X_1$$

$$Y_n = \alpha Y_{n-1} + (1 - \alpha) X_{n-1}$$

for $n = 2, 3, 4, ...$

The parameter α controls the smoothing: high values of α produce heavy smoothing and filter out most random variations.

The current cube acquires an extra dimension, with elements `Absolute` and `Trend`.

See also **FORECAST** for a discussion of the assumptions of trend extraction.

User options

- Smoothing parameter α (between 0 and 1)

Side effects

- The current cube acquires an extra dimension.

Errors and warnings

- Error - The current cube contains less than two periods

Related topics

Topic	Section	Page
COMPUTE	3.3.1	35
FORECAST	3.7.5	82

3.7.5 FORECAST

Function

Automatic extrapolation of timeseries, based on trend.

Motivation

This module appends a number of periods to the current cube and extrapolates all timeseries.

The number of new periods is user-selected. The labels are assigned as `'Extra 1'`, `'Extra 2'` etc.

The module computes a trend over the past and extends it to the new periods.

Given a timeseries X_n, we define the trend as a second timeseries Y_n obtained as

$$Y_1 = X_1$$

$$Y_n = \alpha Y_{n-1} + (1 - \alpha) X_{n-1}$$

for $n = 2, 3, 4, ...$

The parameter α controls the smoothing: small values correspond to sensitive (and less reliable) extrapolation while high values produce slow-varying (and more conservative) extrapolation.

High values may also produce a jump from the latest observed value to the extrapolated series.

A second 'mixing' parameter (in the range 0 to 1) is used to smooth this transition to the extrapolated series: a high value means a very smooth transition.

The extrapolation obtained by this module is an automatic operation—the result must be accepted only after verifying that some basic assumptions (implicit in the technique) are satisfied by the timeseries at hand.

In this case the assumptions are that each timeseries can be modelled as a *random variable with constant or slow-varying mean* and that deviations from the mean in various periods are not correlated.

In other words, the program will attempt to identify a slow-varying trend plus a random noise component.

It is the user's responsibility to check that these assumptions are reasonable in a particular case.

Example

If the series at hand represents the daily production of a workshop, the assumptions may hold and **FORECAST** will produce a reasonable result.

If the series, on the contrary, holds the total produced to-date then one would expect a typical *ramp* behavior and the extrapolation will exhibit an (unlikely) saturation to a plateau.

In such a case one would do well to obtain period values, extrapolate and re-obtain to-date values of the extended timeseries.

♢ Remember also that demanding users, who may not content themselves with a single extrapolation technique, have the whole power of **COMPUTE** at their disposal to express their extrapolation formula.

Note also that the command **BLOWUP** offers a convenient way to copy the seasonal behavior of one year to the next.

User options

- Number of new periods (on dimension 2)
- Smoothing parameter (between 0 and 1)
- Mixing parameter (between 0 and 1)

Side effects

- The current cube grows on dimension 2
- The new labels are marked `Extra` n

Errors and warnings

- Error - There are less than two elements on the second dimension.

Related topics

Topic	Section	Page
TREND	3.7.4	81
COMPUTE	3.3.1	35
BLOWUP	3.4.4	56

3.7.6 MOVINGAVERAGE

Function

Compute the moving average of all timeseries.

Motivation

This module computes the moving average of all timeseries in the current cube and creates a new dimension to store it.

If you request a moving average on N periods, the value of a cell will be the average of the cell itself and the $N-1$ preceding cells.

The first $N-1$ cells of the result will be missing values.

User options

- Number of periods to average (on dimension 2)

Side effects

- The current cube acquires a new dimension with elements `Absolute` and `Moving Avg` N

Errors and warnings

- Error - There are less than two elements on the second dimension.

Related topics

Topic	Section	Page
TREND	3.7.4	81
COMPUTE	3.3.1	35

3.8 Secondary cube management

Motivation

The central idea of Data Analysis is the current cube, which is the result of all commands issued since an initial **RESET** command.

There are situations where the current cube is not sufficient, because

- data from different databases must be merged
- results from intermediate operations must be stacked on top of each other
- some commands require two input cubes or some formulae refer to both cubes.

All this is made possible by the existence of a *secondary* cube, which survives a **RESET** command.

The secondary cube resides in memory (not on file) and is lost whenever you leave TANGRAM.

Current and secondary cube also share the available RAM memory. When their sizes are too large for the available memory, you risk a Memory Full error.

The **SUMMARY** command reports on the size and labels of both cubes.

See also the module **DOCK** in the Administration chapter.

3.8.1 STORE

Function

Append the current cube to the secondary one.

Motivation

If a secondary cube exists, the current cube is appended to it, on a chosen dimension. The element count on the other dimensions must be conformable.

If the secondary cube, for example, has size $20 \times 8 \times 13$ and the current one has size $20 \times 10 \times 13$, then they can be appended on dimension 2, yielding a secondary cube of size $20 \times 18 \times 13$.

If there is no secondary cube, this module simply copies the current cube to the secondary one.

Note that, when appending a 3-way cube to a 3-way secondary cube, you assume that labels on two dimensions are one-to-one synonyms. In fact the current cube labels on two dimensions are lost in the result. If tha labels differ, TANGRAM will issue a warning to that effect.

User options

- The dimension on which the two cubes are stacked (this is irrelevant when the secondary cube does not exist).

3.8.2 GETBACK

Function

Copy the secondary cube to the current one.

The current cube is overwritten, the secondary cube remains unchanged.

3.8.3 RELEASE

Function

The secondary cube is deleted.

3.8.4 SWAP

Function

Current and secondary cubes are interchanged.

Chapter 4

Administration

4.1 General concepts

Overview

This menu groups all database management functions, that is,

1. Database creation and management
2. Data import and data entry
3. Space allocation and management
4. Formulae that apply to the whole database.

The emphasis is on databases that are much too large to be read into memory as a whole.

If this is not your case, you may save yourself the technicalities of points 3. and 4.

That is, you do not need to understand current and maximum element counts: a cube in memory can freely grow, shrink, be reordered etc. and then be saved to file by **SAVEAS**.

Similarly, in a cube of reasonable size, you can compute elements with the Data Analysis menu, then save the computed cube and store your formulae in a procedure. Several problems disappear: **COMPUTE** defines an element on existing ones, so that circular definitions are avoided.

♢ If a database is used by several people, it is recommended that a database administrator be appointed: this person will be responsible for data integrity, back-ups, imports and similar chores.

Some functions of the Administration menu overwrite the current cube left by the Analysis menu. The secondary cube is not affected.

Cube design

The physical creation of a database consists in space allocation and choice of dimension names. Space is allocated for every cell, even before it contains a value.

The creation of a TANGRAM database deserves some forethought and design:

1. Applicability of TANGRAM to your data structure: the basic data structure must be a multidimensional table of numeric data.

   This implies that the choice of an element on each dimension must identify a single cell.

   For example:

Dimension *ACCOUNTS*:	*long-term debt*
Dimension *YEARS*:	*financial year 2005*
Dimension *COMPANIES*:	*ACME Corp.*
associated cell:	*234500 Dollars.*

2. Cube size and element counts.

   The required disk space (for the `.hlm` file) is simply given by 8 times the product of the element counts on all dimensions.

   For example *100 ACCOUNTS* of *200 COMPANIES* over *5 YEARS* will require 800.000 bytes.

   You may choose to allocate a larger space on disk (see maximum element count) typically to save yourself the inconvenience of moving the database to a larger structure when it grows.

   This version of TANGRAM handles databases up to 2 Gigabytes or, equivalently, 250 million cells.

   The database structure file (`.sf`) is normally very small.

3. The element labels.

   You want them to be self-explanatory (to any intended reader), systematic and concise (to produce compact printouts).

   It is generally a good idea to design multi-facet labels that encode several aspects or attributes, so as to use each facet as a selection or sorting key.

   If you have in mind a hierarchy on a dimension, it is best to describe it with adjacent fields, like `'UK Kent Maidstone'` so that the aggregation steps are readily available.

   The reserved characters `'#*?$&"_'` are best avoided in the labels, because they may create ambiguities.

   Dimensions that are needed by different cubes (like Products or Months) are best defined as a standard namesystem and then copied to each cube with **LABEL2DB**.

4. Remember that you can have up to 15 dimensions and that each dimension can in fact contain several facets or the Cartesian product of several (conceptual) dimensions:

   For example:

   ```
   Alfa Omega Budget
   Alfa Omega Revised
   Alfa Omega Actual
   Italmarket Budget
   Italmarket Revised
   Italmarket Actual
   ```

   The module **CROSSOVER** allows you to later move a facet to an existing or new dimension.

5. If you design a 3-way cube, you may prefer the Standard Orientation:

   - Dimension 1: variables, accounts, parameters or measures
   - Dimension 2: time periods
   - Dimension 3: items, observations, individuals in a population.

   If you adhere to this default you do not need to transpose the cube for queries or time-specific operations.

6. If you are designing a very large database (above, say, 50 million cells) then performance and sparsity become an issue. The modules **NEWDB, CROSSDB, BUILDDB** report on cube size and performance before proceeding to create the files.

   - A good thumb rule is to assign dimensions in the order lightest to heaviest (in terms of element count). For example, your cube may have 5 dimensions with 3 10 15 100 200 elements.
   - If your retrieval pattern is to read all elements on a dimension, it is better if this dimension is among the last.
   - If you intend to define Admin formulae on a very large database, the dimensions on which you compute are best placed among the last.

Cube rank: 3-way versus N-way

This section contrasts the advantages of 3-way and N-way cubes.

The world of hypercubes (*alias*: multi-way tables) can be very rich and complex for the end user to handle.

Yet hypercubes (of, say, 3 to 7 dimensions) are often the *natural* model for company data (and also for statistical and econometric data).

These objects are very hard to handle with Relational DBMSs and spreadsheets, mainly because the basic data structure of these tools is bidimensional.

Also, these tools place hard restrictions even on bidimensional operations: try, for example, to reshape a 1000-row 8-column table into a 500-row 16-column one.

♢ TANGRAM's approach is to offer a full set of transformations for *three* dimensional cubes (hence the name **3-way TANGRAM**) with a view to these advantages:

- This structure is easy to visualize and map to planes, rows and columns for display and printing.
- Three dimensions can accommodate three axes that deserve *specific* treatments, like Measures, Periods and Items.
- Three physical dimensions can also represent three groupings of several logical dimensions.

  The unlimited number of facets (*alias*: codes, fields, keys) on the labels of each physical dimension (and the power of the **CROSSOVER** command) allow you to move and regroup the logical dimensions at a later stage.
- Most sparsity problems are simply avoided by joining two logical dimensions that happen to be in a many-to-many relationship and treating them as a single physical dimension.

  If you know, for example, that a customer only buys a small fraction of existing products, you may have a physical TANGRAM dimension containing two keys (`CUSTOMERS/PRODUCTS`) and sparsity is avoided.
- Three dimensions are easily mapped to Excel's rows, columns, sheets. Several 3-way TANGRAM databases of reasonable size can be mass converted to Excel spread-sheets. N-way databases can also be accessed from Excel, but a page at a time.
- Working with cubes of fixed rank helps to keep the dialogue much simpler. The concept of **MERGE** could be extended to merge a 5-way cube with a 4-way one, but at the price of puzzling and abstract questions.

For these reasons, many TANGRAM functions are implemented for three dimensional cubes.

There are situations, however, where you wish to structure your cube as a higher dimensional object and save it as such.

For example, you wish to access it from the TANGRAM-Reader (the Excel add-on) and, once in Excel, you don't have any tool like **CROSSOVER** to separate the grouped dimensions.

As a rule, you increase the dimensionality as the last operation before publishing a higher dimensionality cube.

The typical sequence of operations is

- Prepare your 3-way database.

- Make it an N-way cube with **PROMOTE, SHUFFLE** and **CROSSOVER**
- Issue a **SAVEAS** command to write the cube to a physical database.
- Update the dimension names.

♢ **The range of allowed cube ranks** (*alias*: dimension counts) **is from 3 to 15.**

4.2 Database administration

4.2.1 SELECTDB

Function

Choose the current TANGRAM database.
The previous current cube is lost, the secondary cube is not.

User options

- the file name `.hlm`

4.2.2 NEWDB

Function

Create an empty database.

User options

- The database name `.hlm` (max 50 chars)
- Dimension names, separated by semicolons.
  For instance : `Accounts;Months;Shops;Products`
- Maximum element counts
- Current element counts

Side effects

- The new database is created with missing values, default labels.

Errors and warnings

- Error - Current element count cannot exceed maximum element count
- Error - 15 dimensions is a system maximum
- Error - 3 dimensions is a system minimum

Related topics

Topic	Section	Page
CROSSDB	4.2.3	92
BUILDDB	4.3.4	100
SAVEAS	3.5.1	64

4.2.3 CROSSDB

Function

Create a new database by crossing existing namesystems, that is, placing each of them on a dimension. The cells are assigned missing values.

Motivation

This is an expedient alternative to **NEWDB**, to use reference namesystems to determine database size.

The maximum element counts are set equal to current counts.

If you want to optimize performance, it is a good thumb rule to assign dimensions in the order lightest to heaviest (in terms of element count).

For example, your cube may have 5 dimensions with 3 10 15 100 200 elements.

User options

- The database name `.hlm` (max 50 chars)
- A list of namesystem names, separated by semicolons.

  For instance

  ```
  Accounts;Months;Shops;Products
  ```

  While in **NEWDB** dimension names are free text, here they must represent existing namesystems. If you leave this input box empty, you will be prompted to choose from the list (instead of retyping the names).

Side effects

- The new database is created.

Errors and warnings

- Error - 15 dimensions is a system maximum
- Error - 3 dimensions is a system minimum

Related topics

4.2.4 DOCDB

Function

Document the current database, displaying current and maximum element count, labels and read-only elements.

4.2.5 DELETEDB

Function

Delete a (non current) database. This module will let you delete an unneeded TANGRAM database and release the disk space.

The operation is equivalent to deleting the two files that make up the database with Windows Explorer commands.

User options

- Database name `.hlm`

Errors and warnings

- Error - The current database cannot be deleted.

4.2.6 RESIZEDB

Function

Adjust the current element counts. Create or delete elements.

Motivation

This module lets you re-adjust the current element count in the current database, up to maximum values specified at database creation.

The element count on each dimension can be increased or decreased independently.

In case you *decrease* the element count, (after confirmation) some labels and the corresponding values are lost.

The formulae are re-checked for consistency only when you access the module **UPDFORMULAE**.

If you need to increase a dimension above the predefined maximum, you have to move the whole database to a larger structure with **MOVEDB**.

User options

- New (current) element counts on all dimensions

Errors and warnings

- Warning - The following elements will be deleted :
- Error - Current element counts exceed maximum counts

Related topics

Topic	Section	Page
MOVEDB	4.2.7	94

4.2.7 MOVEDB

Function

Copy a database to a new physical location, altering the maximum element counts.

Motivation

When the initial allocation of space of a database becomes insufficient this module can be used to copy it to a new location.

The maximum element counts can be redefined, the current element counts remain unchanged.

After the successful completion of this module the old database can be deleted and the new one renamed.

User options

- New database name `.hlm`
- New maximum element counts on all dimensions

Errors and warnings

- Error - Maximum element count is less than current element count.

Related topics

Topic	Section	Page
RESIZEB	4.2.6	93

4.3 Database loading

4.3.1 DATAENTRY

Function

Update a database with a data entry session.

Motivation

This module displays the database in a spreadsheet-like grid and lets you update the numeric cells (not the labels).

You choose which dimensions are expanded on the rows and columns. All other dimensions are represented by a combo in which you can choose the current element.

Other grid buttons allow you to copy the whole current page (labels and cells) to the Clipboard, print it in **REPORT** format, change the number of displayed decimal places.

♢ Remember that numeric cells can be copied from other Windows applications via the Clipboard.

Be sure to highlight a source rectangular area not larger than the target area (on the TANGRAM side) and confine yourself to number cells (not text data).

User options

- Dimensions on rows, columns
- Update rights (a single value, like `ON, OFF`) Note that when choosing `ON` you may still be subjected to restrictions on read-only elements.

Side effects

- The current database is updated.

Errors and warnings

- This module has been tested for element counts up to 3000 on each dimension. Larger element counts may produce a Memory Full or System Limit error.

  It is easy to update subcubes of very large databases with **BROWSE**, then **STORE** them and write them back to file with **DOCK**.

Related topics

Topic	Section	Page
PROTECTDB	4.5.2	113
LOADDB	4.3.2	96
BROWSE	3.2.1	26

4.3.2 LOADDB

Function

3-way Load a numeric field from a flat file to the current database.

Motivation

This module loads a numeric field from a flat file to a TANGRAM database.

Every record of the flat file must contain at least four fields, of which three must code the three TANGRAM coordinates and a fourth must contain the corresponding cell.

In other words, the record fields must contain codes that also appear in the database labels.

The loading operation terminates if a record of unexpected length is encountered.

If execution continues to the end, the program reports on the number of processed, accepted and rejected records.

Records may be rejected if one of the three codes is unknown in the database or if the numeric field is invalid.

A file with extension `.xcp` contains the rejected records (with a trailing field that flags the illegal codes).

♢ The number values in the flat file must be in character format (as opposed to binary, float, packed...) and may use either dot or comma to separate the decimal part. The negative sign can either be before or after the number (`-12.33` or `12,33-`).

User options

- Key on the labels of dimension 1
- Key on the labels of dimension 2
- Key on the labels of dimension 3
- Input file name `.txt`
- Record layout (positions of the three keys and the number field). Fields 1,2,3 correspond to keys on dimensions 1,2,3. Field 4 is the number field. Example:

  ```
  2222  111 444433
  ```

Side effects

- The current database is updated
- Output: file of rejected records `.xcp`

Errors and warnings

- Error - Irregular record lengths in flat file - Loading terminated

Related topics

Topic	Section	Page
BUILDDB	4.3.4	100
UNLOAD	3.5.2	65

4.3.3 TRANSFER

Function

3-way Transfer cells (not labels) between 3-way TANGRAM databases.

Motivation

This module allows for the transfer of data cells between physical databases.

This operation is normally reserved to the data administrator.

Data transfers of this kind may be a one-shot operations (like initializing the budget with actual data) or a routine operation (like feeding one database from another).

For example the aggregation of *REVENUES* and *COSTS* databases may be copied to a *PROFLOSS* database and few lines of this may update some cells of the *BALSHEET* database.

This module transfers numeric values only: the values overwrite the older values in the destination database and take their meaning (in terms of labels).

All processing operations (computation, roll-up, scaling,...) must be done before the actual transfer.

The data to be transferred always takes the form of a 3-way subcube that is a subset of both the source and destination database (two 3-way cubes).

Example 1

The specification of the subcube consists of six lists of indices, for example:

database XXX		database YYY	
accounts	3 6 7	lines	106 110 31
months	1 2 3 4 5 6	periods	25 26 27 28 29 30
companies	7 8	corporations	2 1

Note that the copied subcubes must have the same size in the source and destination databases—the databases themselves need not be conformable.

Once obtained these index lists, the module confirms the meaning of the cells in the two databases.

For example:

database XXX		database YYY	
3	Rev. from sales	106	REVENUES
6	Prod. Cost	110	IND. COSTS
7	Overhead	31	FIXED COSTS
1	January	25	JAN 95 ACT.
2	February	26	FEB 95 ACT.
3	March	27	MAR 95 ACT.
4	April	28	APR 95 ACT.
5	May	29	MAY 95 ACT.
6	June	30	JUN 95 ACT.
7	Alfa Omega	2	ALFA OMEGA
8	Pegasus	1	PEGASUS

It is recommended that you check the equivalence between corresponding labels before proceeding to the actual transfer.

♢ The six element lists can, of course, be expressed with the help of the Set Language (See page 127).

An extra keyword `COMMON` is available in this context: it produces a list of elements that have identical labels in the two databases.

If you answer `COMMON` for all six element lists, you only transfer cells that have equal descriptions in source and destination databases.

♢ The module requires that the index lists in the *destination* database do not contain duplicate elements.

Duplicate elements in the *source* database are accepted and this is often a useful technique to replicate an element several times.

Example 2

If you duplicate a month of the source database onto all periods of the destination database, your lists of elements on dimension 2 may look like this:

database XXX		database YYY	
12	Dec 08 Actual	1	JAN 09 BDG
12	Dec 08 Actual	2	FEB 09 BDG
12	Dec 08 Actual	3	MAR 09 BDG
12	Dec 08 Actual	4	APR 09 BDG
12	Dec 08 Actual	5	MAY 09 BDG
12	Dec 08 Actual	6	JUN 09 BDG

♢ The same physical database may serve both as source and destination.

This may be the case when you do a roll-back, shifting all periods back and making room for a new period.

Example 3

You may specify the following element lists:

database XXX		database XXX	
Dimension 1:	`ALL`	Dimension 1:	`ALL`
Dimension 2:	`2 3 4 5`	Dimension 2:	`1 2 3 4`
Dimension 3:	`ALL`	Dimension 3:	`ALL`

Note that extra care is needed to copy a database onto itself because the copying operations are done strictly in the list order.

In this example, element 2 is copied to element 1, *then* element 3 is copied to element 2, etcetera.

A roll-back operation, as the one illustrated, would also require a correct update of the period labels (that are unaffected by this module).

User options

- Source database
- Destination database
- Indices on dim. 1 of source database
- Indices on dim. 1 of destination database
- Indices on dim. 2 of source database
- Indices on dim. 2 of destination database
- Indices on dim. 3 of source database
- Indices on dim. 3 of destination database

Side effects

- The destination database is updated

Errors and warnings

- Error - Element count mismatch on a dimension
- Error - Duplicate elements in the destination database
- Warning - Some elements in the destination database are read-only or will be overwritten by formula execution.

Related topics

Topic	Section	Page
MERGE	3.4.6	59
DOCK	4.3.7	105

Note that direct assignment of the secondary cube to the current cube (**GLOBAL** with formula `xT`) is another way to transfer cells but not labels—and this applies to cubes of any rank. See an exercise of this type in procedure `TRANSFER` in library `example2`.

4.3.4 BUILDDB

Function

3-way Create a new database from a flat file.

Motivation

This module assists you in the step-by-step transformation of a flat file into a TANGRAM database.

The operation is exploratory in nature, meaning that you may decide the final database shape after several attempts.

The whole operation of the module can be broken down into 6 steps:

Step 1 - Definition of record layout. Field names, positions, nature (code or numeric).

Step 2 - Attribution of fields to the TANGRAM dimensions.

Step 3 - Exploratory analysis of the file to determine the database size.

Step 4 - Acceptance of the resulting size (or repetition of Steps 2 and 3).

Step 5 - Database creation.

Step 6 - Loading of the values.

Example

A file contains production orders of an apparel industry (how many suits of each size).

Step 1 - Let the record layout be:

```
bytes  1 -  5   MOD  Model code
       6 - 10   CLO  Cloth code
      11 - 15   COL  Color code
      16 - 20   VAR  Color variant
      21 - 25   WEE  Due date (week)
      26 - 28   SZ46 Suit count of size 46  -numeric-
      29 - 31   SZ48 Suit count of size 48  -numeric-
      32 - 34   SZ50 Suit count of size 50  -numeric-
```

Step 2 - You associate each non-numeric code to a TANGRAM dimension, for example:

```
MOD -> dim. 1
CLO -> dim. 2
COL -> dim. 3
VAR -> dim. 0 = ignore field
WEE -> dim. 2
```

Step 3 - TANGRAM reports that the resulting cube would have:

- 1000 different MOD codes
- 5000 different CLO-WEE code pairs
- 500 different COL codes.

The three numeric fields SZ46, SZ48, SZ50 have the effect of multiplying TANGRAM's *first* dimension by three, so that the resulting database will have $3000 \times 5000 \times 500$ cells or 7500 million cells.

At this point you see the mistake: you are trying to create a huge sparse database, because a very small fraction of all MOD-CLO-COL keys is present in the flat file.

The best course of action is to redefine the dimensions in a more parsimonious way:

```
MOD -> dim. 3
CLO -> dim. 3
COL -> dim. 3
VAR -> dim. 0 = ignore field
WEE -> dim. 2
```

A new analysis of the flat file produces a more acceptable proposal.

For example:

- 6500 different MOD-CLO-COL keys;
- 10 different weeks.

The whole cube will have the more manageable size of 3 suit sizes $\times$ 10 weeks $\times$ 6500 keys or 195000 cells.

If this choice is accepted, the program will proceed to database creation and loading (Steps 5, 6).

To conclude this example, note that the VAR field was not loaded. As a result, all color variants corresponding to the same MOD-CLO-COL keys will be aggregated during the loading operation.

$\diamondsuit$ The performance of this module may become a critical factor in case of very large files extracted from the company info system.

These considerations can help the administrator to improve the load times:

- The most important factor is the sorting order of the input file. The best situation is when it is sorted on those keys that become the second and third TANGRAM dimensions.
- The available RAM memory is another key factor.
- The record length and the number of fields have little influence on loading times.

- See Main menu, Configuration, Miscellany, for a way to tune the performance of this module. Basically, you control the number of records that TANGRAM processes at a time.

♢ The number values in the flat file must be in character format (as opposed to binary, float, packed...) and may use either dot or comma to separate the decimal part. The negative sign can either be before or after the number (`-12.33` or `12,33-`).

♢ The definition of all inputs for this operation is non trivial. It is best to fill only the first two inputs (the file names) and let the system guide you through the other choices.

If you save this in a procedure it will be clear how to update the resulting strings for minor adjustments.

User options

- Name of input file `.txt`
- Name of new database `.hlm`
- Record layout (a mask of n field positions)
- Field names (a list of n field names, separated by commas)
- Number fields (a Boolean string of n values, where 1 marks number fields, 0 code fields)
- Destination of fields in the cube (a string of n integers in the range 0,1,2,3 —where 0 means that the code field is to be ignored— 1,2,3 that it must become part of this cube dimension).

Side effects

- Database creation and loading
- Byproduct: three namesystems (named `00001, 00002, 00003`) contain the cube labels on the three dimensions. These may be useful to analyze the contents of the flat file even without creating the physical database. The three namesystems are overwritten at each file analysis.

Errors and warnings

- Error - No numeric field in record layout
- Error - Irregular record length in the flat file. Loading terminated.
- Warning - In n cases a number field contained an ill-formed number.

Related topics

If you contrast **BUILDDB** with **LOADDB** you notice that the former:

- does not require a pre-existing database
- loads both codes and values in one operation
- accepts records with several numeric fields and more than three codes
- aggregates while loading, which means that values corresponding to the same keys are not overwritten but added up.

4.3.5 FILLDB

Function

Fill the current database with test data, obtained from the marginal values.

Motivation

TANGRAM is often used to produce prototype applications in very short time scales. This module offers a sophisticated way to fill a cube with likely test data.

The marginal values used to fill the whole cube are the cube edges. In a 3-way cube, for example, they are at these coordinates:

- Dimension 1 : `ALL`
  Dimension 2 : `1`
  Dimension 3 : `1`

- Dimension 1 : `1`
  Dimension 2 : `ALL`
  Dimension 3 : `1`

- Dimension 1 : `1`
  Dimension 2 : `1`
  Dimension 3 : `ALL`

♢ The module **FILLDB** fills in all other cells in proportion to their marginal values.

You can think of this as dividing each marginal string by `T[1;1;1]` and then multiplying these strings in all possible combinations. The corner cell `T[1;1;1]` must not be zero.

In mathematical notation, the formula for a 3-way cube would be

$$T_{ijk} = T_{111} \frac{T_{i11}}{T_{111}} \frac{T_{1j1}}{T_{111}} \frac{T_{11k}}{T_{111}}$$

After obtaining a *base value* in this fashion, the program adds to it a random value with maximum range h percent of the base value.

For instance, if the base value of a cell is 1200 and $h = 10$ then the random component will be uniformly distributed between -120 and $+120$.

If you choose $h = 0$ (no random component) then all cells will be strictly proportional to marginal values.

♢ It should be evident that this module helps you to fill huge cubes with very little direct data entry on your part. If the cube is $100 \times 12 \times 50$, you enter only 160 numbers and automatically produce 60000 cells.

These values will exhibit the same seasonal behaviour of the marginal data and also reflect the relative size of the elements on all dimensions.

User options

- Random component (as a percentage of base value)

Side effects

- Implicit input: marginal strings in the database
- All cells of the current database are overwritten (including the marginal strings, if $h > 0$).

Errors and warnings

- Error - Marginal strings contained missing values

Related topics

Topic	Section	Page
RANDOMDB	4.3.6	104

4.3.6 RANDOMDB

Function

Fill the current database with (uniformly distributed) random data.

Motivation

This module fills the entire database with random values.

Every cell is assigned a random value, uniformly distributed between two user-supplied limits.

User options

- Minimum value
- Maximum value

Side effects

- The current database is totally overwritten.

Related topics

Topic	Section	Page
FILLDB	4.3.5	103

4.3.7 DOCK

Function

Write the *secondary* cube to the current database, provided all its labels already appear among the database labels.

Motivation

This is one of the ways to update a database or transfer cells from a different database that shares (a subset of) the same labels.

If a single label of the secondary cube is not found on the corresponding dimension of the current database, the operation is not carried out.

The dimension names are irrelevant in this operation.

Side effects

- The current database is reset, losing the current cube.
- The current database is overwritten.

Related topics

Topic	Section	Page
SAVEAS	3.5.1	64
TRANSFER	4.3.3	97

4.4 Administration formulae

4.4.1 UPDFORMULAE

Function

Define or modify the formulae on a given dimension of the current database.

Motivation

The Data Analysis menu offers very sophisticated computations.

Such computations are, in practice, private ones, that is, they are defined, saved and re-used by each user of a database.

The Administration menu offers slightly different features, for the kind of computations that are defined, saved and re-executed by the administrator for the benefit of all users.

The module **UPDFORMULAE** lets you define independent computations on each TANGRAM dimension.

You may then have several *packets* of formulae which are separately defined and applied.

In the simple cases, computations are defined on the first dimension only (for example, on *Variables*).

If you define computations on several dimensions you must be aware that the order of execution does matter.

Example

If your database contains *VARIABLES*, *YEARS*, *COMPANIES*, you may want to compute ratios (on *VARIABLES*) and company consolidations (on *COMPANIES*).

It is easy to see that you want first to consolidate companies and then to compute the ratios.

If you do it the other way around, the ratios of the consolidated company will be the sum of the individual ratios.

♢ The general rules for coding the formulae are the same as for **COMPUTE**, with a few notable exceptions:

- All formulae that act on a given dimension are collected in a table (one per line)
- Every formula must assign its result to a given element (with a colon symbol).

Examples

```
[33] : 100 TIMES [10] DIVIDE [45]
"Margin" : "Cost of Goods" LESS "Overhead"
```

- The keywords

  ```
  PRE POST SLIDE todate toperiod STAT PERCENTILE INLIST SUB
  ```

  are not supported here. You may, however, mix the set language and the computation keywords.

  Examples

  ```
  [24] : [22] DIVIDE SUM [SPOT '2001']
  [30] : (SUM [ ALL BUT 21 TO 30] ) DIVIDE SUM [21 TO 29]
  ```

- Each element can be referenced either by its index within brackets (for ex. `[131]`) or by the double-quote notation (for ex. `"Variable costs"`). The string between double quotes must identify one and only one element among the labels of the chosen dimension.

- The packet of formulae on a given dimension must not contain circular definitions.

  In other words, TANGRAM must be able to compute the elements in some order.

Error example 1

```
[7] : [3] PLUS [5] PLUS [1]
[5] : [2] DIVIDE [4]
[4] : [7] TIMES [20] TIMES [21]
```

Error example 2

```
[26] : [26] TIMES 1.15
```

Example 3

```
[7] : [3] PLUS [5] PLUS [1]
[10] : [2] DIVIDE [4]
[2] : [1] TIMES 36500
```

As you note in **Example 3**, the formula that defines an element may refer to elements above and below it.

The physical order in which elements appear in the database is irrelevant.

This implies that you will never have to physically move elements to make room for a new addendum or to adjust element indices in a formula.

Finally, the order in which you enter your formulae is irrelevant. In the last example we have defined elements *7, 10, 2* in that order.

If the formulae cannot be computed sequentially in the way you enter them, TANGRAM will re-order them to reflect the sequence of execution.

Example 4

You may find the three formulae of **Example 3** reordered as follows:

```
[2] : [1] TIMES 36500
[7] : [3] PLUS [5] PLUS [1]
[10] : [2] DIVIDE [4]
```

Since these formulae define elements *2, 7, 10*, these elements will become read-only elements.

♢ When you update the formulae, TANGRAM will check their consistency before you are allowed to leave the editor and exit.

The final check for correctness is carried out by a different module **CHECKSYNTAX**.

♢ If the labels on the chosen dimension are an *indented namesystem* then TANGRAM will volunteer to convert the indentations to the corresponding formulae. Automatically generated formulae are marked with the comment `! INDENT` .

When this is done, you can still edit the packet of formulae and enter your own.

Example 5

If the labels contain:

```
Europe
___Germany
___United Kingdom
______Scotland
______England
___France
___Italy
______Tuscany
_________Florence
```

then the generated formulae will be

```
[3] : SUM [,4 5]       ! INDENT
[8] : SUM [,9]         ! INDENT
[7] : SUM [,8]         ! INDENT
[1] : SUM [,2 3 6 7]   ! INDENT
```

User options

- Choice of the dimension.
- An editor session to enter or update the formulae.

Side effects

- Formulae are re-ordered
- The formulae are stored as part of the current database
- Elements assigned in the formulae are defined as read-only.

Errors and warnings

- Error - Mismatched parentheses
- Error - Mismatched brackets
- Error - A line must contain one and only one assignment (colon)
- Error - Illegal use of the double quote notation `" "`
- Error - This formula is part of a circular definition
- Error - An element was defined more than once
- Error - Formula refers to non-existing elements

Related topics

Topic	Section	Page
Available keywords	3.3.1.1	39
COMPUTE	3.3.1	35
GLOBAL	3.3.4	47
GROWTREE	3.3.3	45

4.4.2 CHECKSYNTAX

Function

Check the syntax and validity of the administrator formulae.

Motivation

This module actually executes the formulae on dummy data, to guarantee that they can be safely applied to the database.

The normal output of the module is the message

```
The syntax of all formulae is correct.
```

or else a list of offending formulae and the nature of the problem.

For example:

```
Dimension Variables
Formula N.3

[2] : [1] DEVIDE [3]      NAME UNKNOWN
```

This verification is more thorough than the one performed by the module **UPDFORMULAE** since the latter performs a formal and partial verification, while **CHECKSYNTAX** actually executes the formulae and intercepts all errors.

User options

- Choice of the dimension.

4.4.3 SHOWFORMULAE

Function

Spell out the labels of all elements referred to in a formula.

Motivation

This module is used to display the full labels of all elements that appear in a formula, even if they are referenced by index or by the double quote notation.

It is a good practice to document and print the whole packet of formulae for desk checking.

Example

The formula

```
[39] : ( SUM [32 TO 36] ) DIVIDE "Tot overh"
```

will produce an output such as:

```
Meaning of the variables:
   [39]   Revenues/Overhead
   [32]   Sales
   [33]   Commissions
   [34]   Interests
   [35]   Royalties
   [36]   Maintenance
   [10]   Total overhead
```

User options

- Choice of the dimension.
- Elements to analyze.

4.4.4 FAMILYTREE

Function

Trace the elements that indirectly influence a computed element.

Motivation

When the chains of computations are long and complex it is useful to trace the input elements that produce a result—in TANGRAM jargon the *family tree* of the element.

The dependency is highlighted by the indented margin.

Example

```
[ 38]  Total costs
   [ 19]  Fixed overhead
   [ 37]  Material costs
      [ 10]  Cabinets
      [ 11]  CPU
      [ 35]  Disks
         [ 12]  Removable disks
         [ 13]  Fixed disks
         [ 14]  Optic disks
      [ 36]  Labour costs
         [ 15]  Salaries
         [ 16]  External contracts
         [ 17]  Travel and expenses
         [ 18]  Consultants
```

User options

- Choice of the dimension.
- Elements to analyze.

4.4.5 RUNFORMULAE

Function

Recompute the current database.

Motivation

This module recomputes the current database applying administrator formulae.

The computation can be applied to several dimensions in a single execution (thus reading the database only once).

If you define formulae on several dimensions, do remember that the precedence does matter.

Be sure to recompute the database whenever you have used one of these modules:

- **LOADDB**
- **TRANSFER**
- **DATAENTRY**
- **UPDFORMULAE**

User options

- Dimensions to recompute

Side effects

- The current database is updated

Errors and warnings

- Any interpreter-level error due to illegal formulae
- Warning - No formula is defined on the chosen dimension

4.5 Database labels

4.5.1 UPDDBLABELS

Function

Update element labels and dimension names.

Motivation

At database creation, labels are defined by TANGRAM as a numbered list. The first operation you want to perform is the definition (or import) of meaningful labels, that will assist you in all subsequent operations.

Note that a label is associated to the physical position of the cell in the file.

If the administrator shifts all labels of a dimension down one line, this will not move data cells on disk but will wrongly label all data in the database!

For similar reasons, you may not change the label count while editing: see **RESIZEDB** for this purpose.

User options

- Dimensions to update. The table of dimension names is treated as appended to the other labels, so that, in a 4-way database, you may code `1 2 3 4 5` to update all tables.

4.5.2 PROTECTDB

Function

Redefine read-only elements in the current database.

Motivation

TANGRAM has a notion of protected cells, in which the numeric value cannot be manually updated.

Normally, only computed cells are write-protected. In other words, TANGRAM analyzes the formulae on every dimension, determines which elements are updated and defines such elements as read-only.

The reason is, of course, that such values will be overwritten whenever you recompute the database.

In some cases you want to extend this protection to other elements, so that you do not inadvertently update "actual" values.

In other cases, you want to lift the protection and update computed elements for a simulation run (this may be quicker than updating all the detail elements and recomputing).

This module lets you redefine protected elements on each dimension.

A cell is protected if one of its coordinates is a protected element.

This means that you are not allowed to define protected cells on a cell-by-cell basis, as you can do in **BROWSE**.

Remember that when you execute the module **UPDFORMULAE** protected elements are re-assigned to coincide with computed elements.

The read-only elements are tested only by **DATAENTRY**. Administration services like **LOADDB** ignore this setting.

User options

- Dimension
- List of read-only elements

Side effects

- Current database attributes are updated

Related topics

Topic	Section	Page
UPDFORMULAE	4.4.1	106

4.5.3 DB2LABEL

Function

Copy the current database labels to a namesystem.

Motivation

This module copies (or appends) the database labels to a namesystem.

User options

- Source dimension
- Append mode (a Boolean value. Use 1 to append the labels as extra rows in an existing namesystem. Use 0 to create a new namesystem).
- Destination namesystem (existing or new, according to the Append mode)

Related topics

Topic	Section	Page
LABEL2DB	4.5.4	114

4.5.4 LABEL2DB

Function

Copy a namesystem to the labels of the current database.

Motivation

This module copies (or appends) a namesystem to the labels of the current database. The namesystem can be appended to the existing labels only if the resulting element count does not exceed the maximum count.

User options

- Source namesystem
- Append mode (a Boolean value. Use 1 to append the namesystem to the labels and expand the database. Use 0 to overwrite the labels).
- Destination dimension.

Errors and warnings

- Error - The resulting element count would exceed the maximum.

Related topics

Topic	Section	Page
DB2LABEL	4.5.3	114

Chapter 5

Namesystems

Overview

Namesystems (*alias*: labels, nomenclatures, descriptors) are two-way character tables in which every row describes an element.

They are the same kind of objects as the labels in a TANGRAM database—in fact, namesystems are created in view of their use in databases.

A namesystem describes the elements of some collection, such as Branches, Suppliers, Accounts, Companies, including all facets (*alias*: fields, codes, keys, properties, attributes) that pertain to them.

Example 1

Suppose this is a list of companies, entailing several facets: code, description, state, region, country and size:

```
8762 Alfa Omega     CA EAST   USA Large
5266 Pegasus        TX SOUTH  USA Small
2992 Trading Intl.  NY WEST   USA Medium
7382 Three Oaks     TX SOUTH  USA Small
```

♢ Remember that you do not formally declare these facets in TANGRAM—you simply point at them when required. For this reason fixed-width facets are used.

There is no preset limit to the number of rows, the number of facets and the total width of a namesystem.

The order of the rows carries no special meaning but a namesystem is not supposed to have duplicate rows.

If a namesystem contains duplicate rows, you get a warning with a list of lines that are not unique (all occurrences of duplicate rows, except the first from top).

The same type of error is issued for cube labels, if a module requires unique labels.

♢ Namesystems reside in TANGRAM libraries `.lib` . They are permanently saved when the current library is saved (or when you leave TANGRAM).

They can be created in several ways:

- Via an Editor session
- From a text file
- From elementary namesystems, via the four commands **SYNONYM, CROSS, CONVERGE, PAIR** described in this chapter.
- From database labels
- By executing **BILDDB**. See the namesystems `00001 00002 00003` that are left (and overwritten) by each execution.

Namesystems typically contain:

- codes, descriptions and attributes for *join* operations (typically used to expand database labels that contain only codes).
- a master copy of labels used in several databases.
- a master list of elements (like Products and Customers) of which database labels must be a subset.
- elementary namesystems that are used to generate more complex ones, for example with the **CROSS** command.
- aggregation paths.

Indented Namesystems

One of the meanings of the facets in a namesystem is to specify a hierarchy or aggregation path.

The companies in Example 1, for instance, can be aggregated on State, Region, Country or Size.

When there is a single aggregation path and the aggregation tree has uneven depth, TANGRAM also supports a different technique to define a hierarchy—this is called an *indented namesystem.*

In this notation an element is defined as the sum of all lower elements that have a deeper indentation. Each indentation step is marked by a consistent number of underscores (for example, 3 underscore characters).

Example 2

```
Europe
___Germany
___United Kingdom
______Scotland
______England
___France
___Italy
______Tuscany
_________Florence
```

When such a namesystem is used in a database, TANGRAM generates the appropriate formulae and read-only attributes.

Alternatively, the database labels may contain only the atomic elements and the aggregations are produced by the **GROWTREE** module.

♢ This chapter groups all modules that are used to maintain a population of namesystems.

Related topics

Topic	Section	Page
DB2LABEL	4.5.3	114
LABEL2DB	4.5.4	114
CROSSOVER	3.4.2	53
LABELJOIN	3.1.6.2	25
UPDFORMULAE	4.4.1	106
Indented tables	6.5	133

5.1 NEWLABEL

Function

Create a new namesystem via the editor.

An acceptable name consists of upper- and lower-case letters, digits and underscores (max. 25 char).

Example:

```
PROFIT_AND_LOSS_ACCOUNTS
```

Remember that names are case sensitive.

User options

- the namesystem name.

5.2 UPDLABEL

Function

Open an editor session to update an existing namesystem.

User options

- the namesystem name.

5.3 DELETELABEL

Function

Delete a namesystem in the current library.

Note that erroneously deleted namesystems can still be recovered from the copy of the library on disk.

User options

- the namesystem name.

5.4 LISTLABEL

Function

List the names and sizes of all namesystems in the current library.

5.5 DOCLABEL

Function

Document the namesystems in the current library.

User options

- list of namesystems to include in the documentation.

5.6 TXT2LABEL

Function

Import a text file that becomes a namesystem.

User options

- source file `.txt`
- append mode (Boolean - Use 0 if the namesystem does not exist - Use 1 if it exists and the file must be appended as extra rows)
- destination namesystem.

5.7 LABEL2TXT

Function

Export a namesystem that becomes a text file.

User options

- source namesystem
- append mode (Boolean - Use 0 if the file does not exist - Use 1 if it exists and the namesystem must be appended as extra rows)
- destination file `.txt`

5.8 SYNONYM

Function

One-to-one relationship between namesystems.

Motivation

The meaning of this operation is to declare that the two namesystems are two forms of the same classification.

For example, the two namesystems `COMPTES` and `ACCOUNTS` may define the same accounts in French and English.

The effect is simply to create a two-field namesystem, appending the two input namesystems side-by-side.

User options

- left namesystem
- right namesystem
- result namesystem

5.9 CROSS

Function

All-to-all relationship (*alias*: Cartesian product) between namesystems.

Motivation

Produces a two-field namesystem containing all combinations of the rows of the component namesystems.

Example

From a 5-element product list and a 4-element color list, you would get $5 \times 4 = 20$ combinations:

Radiator
Fan
Conditioner
Humidifier
Oven

White
Blue
Yellow
Red

Radiator	White
Radiator	Blue
Radiator	Yellow
Radiator	Red
Fan	White
Fan	Blue
Fan	Yellow
Fan	Red
Conditioner	White
Conditioner	Blue
Conditioner	Yellow
Conditioner	Red
Humidifier	White
Humidifier	Blue
Humidifier	Yellow
Humidifier	Red
Oven	White
Oven	Blue
Oven	Yellow
Oven	Red

One application of **CROSS** is to combine two or more namesystems that will reside on a single TANGRAM dimension.

The **CROSSOVER** module, in Data Analysis, will move one of these facets to another dimension.

User options

- left namesystem
- right namesystem
- result namesystem

5.10 CONVERGE

Function

Specify the many-to-one relationship between two namesystems.

For example, you want to define how Products are grouped into Product Lines.

User options

- left namesystem
- right namesystem
- result namesystem

5.11 PAIR

Function

Specify the many-to-many relationship between several namesystems.

For example, you have master namesystems of Products, Colors, Sizes and want to define which triplets are actually produced by your company.

This version of **PAIR** accepts from 2 to 7 constituent namesystems.

User options

- result namesystem
- line count in the result namesystem
- input count (number of fields in the result)
- input number 1 (a namesystem name)
- input number 2
- input number 3
- input number 4
- input number 5

- input number 6
- input number 7

Errors and warnings

- Warning - Duplicate rows in the resulting namesystem

Chapter 6

User interface and keywords

This chapter lists the GUI (Graphical User Interface) modules and explains the Set Language keywords for element selection.

The following sections list other keywords and utility programs for advanced uses of TANGRAM.

6.1 Dialogue boxes

The interaction with the TANGRAM user is handled by eleven general purpose modules that are listed here.

Their behaviour is, in general, what a Windows user expects.

Some differences and extra features are illustrated here.

1. Questions and Answers

   This module handles from zero to ten questions and answers, displays error messages and loops until it obtains acceptable answers (or the user hits "Cancel").

   The main idea is that you may either formulate your answer via keywords (and even APL expressions) or leave the input field blank to be prompted by an appropriate GUI module.

   - If the answer is numeric, you may simply type in any expression that produces an acceptable result. For instance, instead of typing element indices as

     ```
     1 2 3 4 5 6 7 8
     ```

     you may use the more general reply

     ```
     1 TO COUNT 3
     ```

     or you may also assign a name to this answer so that you later reference it:

     ```
     MyList: 1 TO COUNT 3
     ```

A minus sign next to a constant is interpreted as the sign; if it is not adjacent to a constant it performs a difference. For example, `10 -8` produces a string of two integers, whereas `10 - 8` produces the result 2.

If you use several keywords, remember that they are evaluated right to left, unless you use parentheses.

For example: `( T > 200 ) OR T < 100`

- If the expected answer is text, it is taken at face value. If you wish to enter an expression instead, start it with character `'='`.

  For example, when you enter a page title and want to assemble it from your own variables, you may enter something like:

  ```
  = 'Department of ',userOffice,'  Printed on ',TS
  ```

  For example, when you enter a file name you may want to assign it in a systematic way with a reply like:

  ```
  = ProjectPath, Prefix, SeqNumber, '.hlm'
  ```

- If you leave some input fields empty and hit "Enter", TANGRAM will volunteer some context-dependent assistance.

  For example, you may be selecting elements and leave the list empty. TANGRAM will open a Line Selection Box and let you choose from it.

  The collection of selected element indices will be stored in the procedure.

  To choose an empty element list, use the keyword `NONE`.

♢ Note that the procedure stores the actual answers and keywords as you type them. If, instead, you intend to save the *current result* of keyword execution, assign it to a variable.

2. Line Selection Box

   According to the context, this can be a single or a multiple line selection.

   Left-click selects a single element

   Shift Left-click extends the selected range to this element

   Ctrl Left-click adds a single element to the selection or de-selects an element.

3. Column Selection Box

   This operation is peculiar to TANGRAM. It consists in choosing columns, that is, character positions within a displayed character table (for example, pointing to fields in a file, to facets in a label).

   According to the context, this can be a single or a multiple column selection. In the former case, you have four pushbuttons available (Select

cols, Deselect cols, Fill rest, Deselect all), in the latter you have two more (Next marker, Prev. marker).

How to proceed:

- You are shown the top rows of your table. Highlight a rectangular area by dragging the left mouse button on the top row of grey tiles.
- Click on "Select cols" to transform the highlighted area into your selection.
- Click on "OK" to close the window with a single field selected,

Or, in case of multiple fields:

- Repeat for other sets of columns, to define the next fields.
- Click on "OK" when finished.

The mask you are defining will be visible as a string of characters, like `2222 111111133331111` where ones marks the first field, twos the second and so forth. The marker is changed (from '1' to '2' etc.) whenever you select some columns. You can also change it manually with buttons 'Next marker' 'Prev. marker'.

4. Binary choice

   This module displays a question and expects a Yes/No answer.

5. Information

   This is used to display text of five kinds (information, warnings, errors, program failures, printout pages and comments).

   You simply hit the Enter key or click on "OK" when you are ready to proceed. The button "Print" sends the whole text to the standard printer.

   In the case of printout pages, you may also click on "Cancel" to stop producing the rest of the printout. Note that if you are sending a printout to screen, printer and file, the "Cancel" action will apply to all of them.

6. Editing a character table

   The editor is used to create or update character tables (in particular labels, namesystems, formulae, procedures).

   The editor adheres to the Windows standard.

   In particular:

   - Esc concludes the editing session
   - Shift Esc quits (aborts editing, leaving the original table unchanged)
   - Shift Del Cuts the highlighted area and copies it to clipboard
   - Ctrl Ins Copies to clipboard

- Shift Ins Pastes from clipboard
- Del Deletes the highlighted area
- Ctrl Shift Ins Opens a line at the cursor
- Ctrl Del Deletes the current line.
- - Key-pad minus toggles line numbering. Note that the editor numbers the lines from zero.

7. Progress bar

   Displays the progress of a time-consuming operation. After some time, it also estimates the time to finish, in the form `hh:mm:ss`

   This window is on top of all other windows and applications.

8. Grid

   This is the standard form for data entry and cube display. Two dimensions are expanded (and scrollable). The others are represented by combos where you can pick a different element.

   You can also use Copy-and-Paste with other Windows applications.

   The toolbar contains the following buttons:

   - "OK" Leaves the grid, updating the values.
   - Printer icon. Prints the current page with the **REPORT** module and default options.
   - Clipboard icon. Copies the whole current page (labels and cells) to the clipboard.
   - ".0 .00" Displays one more decimal place, with a maximum of 9
   - ".00 .0" Displays one decimal place less, with a minimum of zero.

   The characters used for comma and thousand separators are taken from the Windows settings (Settings, Control Panel, Regional Options, Numbers).

9. File Selection Box

   Selects a file name with a given extension.

   Two modes available: existing file or new file.

10. Menu

    This is the standard menu. It lets you move on a tree-structured option list.

    A different icon (traffic light) marks available items, unavailable items and item groups.

    To select an available item, either double click on it or highlight it and press Enter.

To quickly move to an item, enter the first letter(s).

In all menus (except the Main menu) upper-case item names correspond to actual program names (that you also find in procedures).

To see what an item does, highlight it and see the hint.

To leave a menu, hit Esc or click on "Cancel". This will bring you back to the parent menu.

11. Display elements box

    This dialogue box displays a collection of element names (like the procedure names) in a list box and also displays the contents of the current element (like a procedure body).

6.2 Set Language

This section illustrates various keywords to select the elements of a set (typically the elements of a cube dimension).

The keywords introduced here, taken together, constitute a *Set Language* that complements mouse-oriented selections.

♢ Your reply may reference the elements:

- via their indices in the set, for example:

  ```
  12 TO LAST
  ```

- via their labels, for example:

  ```
  ANY 'BUDGET'
  ```

- via the associated cell values, for example:

  ```
  WHERE ALLTRUE T > 100
  ```

  This last feature applies only to Data Analysis and requires the cube to be real.

To wrap it up, the Set Language can be a free combination of available keywords.

Examples

The simplest examples should be self-explanatory:

```
"First margin"   TO  "Overhead"
100  TO 121, 128  95, 130  TO  111
LAST  TO  LAST - 5
(LAST - 5)  TO  LAST
6, ALL  BUT  SPOT 'Pesetas'
ANY 'December'
ALL BUT ANY 'Ratio'
AZ BUT LAST
```

The rest of this section presents these keywords in detail.

- Keywords that stand for element indices.

Keyword	Meaning	Example
`NONE`	no elements	`NONE`
`ALL`	all elements	`ALL BUT 17 19`
`FIRST`	the first element in the set	`LAST TO FIRST`
`LAST`	the last element	`7 TO LAST`
`EVERY` *n*	take one element every *n*	`EVERY 7`
n `TO` *m*	the integers from *n* to *m* inclusive	`20 TO 10`
lA `BUT` *lB*	the elements in the list *lA* except those in *lB*	`ALL BUT 22`
n / *m*	replicates *n* times the value *m*	`10/1`

- `ANY` *mask* —Selection on fixed label positions

  The keyword `ANY`, followed by one or more character strings, lets you select all elements that have the character strings in the positions of the label. Quotation marks in the mask represent dont-care positions.

  `ANY 'P4331' '?????JAN'` will select all labels that have `P4331` in the first five positions, followed by those that have `JAN` in positions 6 7 8 (possibly with repeated elements).

  Note that `ANY` is case-sensitive (`ANY 'abcd'` is different from `ANY 'ABCD'`).

  See also `BYKEY` for an alternative.

- `AZ` —Sort the labels alphabetically.

  The keyword `AZ` returns the element indices that sort the labels (A-to-Z). Note that `AZ` is case-insensitive.

- —Double quote notation

  This notation selects the elements via non-positional character strings.

  Example:

```
"TX  BDG"  "CA  BDG"  "NY  BDG"
```

  Each search string (between double quotes) is accepted if it identifies a single element in the set. If the search string contains words or strings separated by blanks, as in `"TX  BDG"`, there must be one and only one element that contains both `"TX"` and `"BDG"`.

  For example the label may be `"BDG  Revenue  TX  1995"`.

  Note that this notation is case-sensitive but position-independent.

- `SPOT` *mask* —String search

  This keyword returns the indices of all labels that contain given words (or strings) anywhere in the label and in any order.

  Unlike the double quote notation, this keyword accepts a multiple result (any index count).

  Unlike `ANY` , it does not specify the string position within the label.

  Example:

  ```
  SPOT 'budget Germany  2005'
  ```

- *tabA* `index` *tabB* —Matching rows in two tables

  Returns the indices of the rows of *tabB* in *tabA*.

  If *tabA* has a row count of N then the result will contain $N + 1$ for the rows of *tabB* that are not in *tabA*.

- `MAJOR` *Percent* —Collection of largest elements

  Selects the largest elements on a cube dimension to obtain up to *Percent* of the total.

  This keyword (used in reply to **SELECT**) will ask you to pick a single element on all other dimensions, so as to define a single string of cells.

  These cells will be chosen in decreasing order, stopping before exceeding the specified *Percent*.

  The final result of `MAJOR` is a list of element indices on the chosen dimension.

  Example

  Your current cube contains Variables, Periods and Customers. You want to extract a list of Customers that make up 80 percent of the Revenues in Total 2007.

  You select on Customers and reply `MAJOR 80` as the list of elements.

  The keyword `MAJOR` asks you to specify a Variable and a Period and you choose Revenues and Total 2007.

  The final result is a cube that contains the largest Customers that constitute no more than 80 percent of the total.

  To select the 10 largest Customers instead, you simply select with `BYVALUE` and then select again `1 TO 10` .

- `WHERE` *Condition* —Transform a Boolean condition into indices

  This keyword transforms a Boolean condition into element indices.

  For example `WHERE 0 0 1 0 1 1 1` produces `3 5 6 7`

For example `WHERE T < 0` returns the coordinates where T is negative: if the cube is 3-way, this is a table with three rows and as many columns as there are negative values.

Typical uses of `WHERE` reference the current cube `T` or the secondary cube `xT`

Examples

```
WHERE ALLTRUE T < 0
WHERE ALLFALSE (T = ND) OR (T = 0)
WHERE T[1 ; 3; ] < 55.5
WHERE ALLTRUE T > xT TIMES 1.2
```

- `BYBREAK ''` —Select elements where a key changes

  This keyword will activate a Column Selection Box to choose the key and then return all indices where the key changes.

  If your labels are :

  ```
  Italy   Rome
  Italy   Milan
  France  Paris
  Germany Frankfurt
  Germany Berlin
  Germany Bremen
  ```

  and you choose the Country facet as your key, then `BYBREAK ''` returns `1 3 4`

  Typical use: flag the rows to underline (in **REPORT**).

- `BYKEY ''` —Step by step selection on several keys

  This keyword will activate a Column Selection Box to enter a multiple key definition and then let you choose from the unique contents of each key.

  In the Country-City example of `BYBREAK` , you may have a long list of cities, where you want to choose first on the countries and then on the individual cities.

  In this way every field becomes searchable but your result is still expressed as indices of the cube elements (that is, in the form that a module like **SELECT** expects).

- `BYLABEL ''` —Sort the elements on several keys

  This keyword will activate a Column Selection Box to enter multiple sorting keys and then sort the current elements on the labels.

  If the current labels contain three facets Product-Color-Size, for instance, you may want to sort on Color, then on Product.

- `BYVALUE` —Sort the elements on corresponding cell values.

  This keyword lets you sort the elements on increasing or decreasing cell values.

  If you are selecting on a dimension, you have to pick an element on each of the other dimensions to identify the sorting string. The module will activate several Row Selection Boxes to enter these choices and then return the element indices that satisfy this criterion.

  You can think of this as freezing all coordinates except one, to identify the string of values that will be used for sorting the whole cube.

- `PICK` —Get the element list at each execution.

  If you leave an element list empty, you will be prompted for your choice by a Row Selection Box and your choice will be stored in the procedure.

  Using the keyword `PICK` instead, means that you want to select a different element list at each procedure execution.

  The keyword `PICK` stands for the user-chosen element list.

  As such, it can be combined with the rest of the Set Language.

  Examples

  Let me choose each time:

  ```
  PICK
  ```

  Let me choose, but elements starting with 'Actual' are forbidden:

  ```
  PICK BUT ANY 'Actual'
  ```

  Let me choose and remember this list:

  ```
  MyChoice: PICK
  ```

  Take all elements, except those that I choose:

  ```
  ALL BUT PICK
  ```

- `ToJoin` —Match the labels of the seconday cube.

  Advanced This keyword, used in **SELECT**, picks the elements so that they match the corresponding labels in the secondary cube.

  If all labels find a match, it aligns the two cubes on an axis.

  See procedure `JOIN_BETWEEN_CUBES` in library `example3` for details.

6.3 Boolean operations

This section lists the keywords for Boolean (*alias*: logical) operations.

Keyword	Meaning	Example
x `APPROX` y	True if $x = y$ to within 0.01%	`T APPROX xT`
x `AND` y	True if both x,y are true	`(T>0) AND xT=0`
x `OR` y	True if x or y is true	`(T>0) OR T=0`
`NOT` x	True if x is false and viceversa	`NOT T=ND`
`ALLTRUE` y	True if y is true everywhere (in all cells of the other dim.)	`ALLTRUE T > 100`
`ALLFALSE` y	True if y is false everywhere	`ALLFALSE T > 0`
`WHERE` y	Indices of ones in Boolean y	`WHERE ALLTRUE T>0`
`ON`	Boolean 1 (for readability)	`ON,OFF,OFF`
`OFF`	Boolean 0	`OFF`

6.4 Set operations on tables

These are utility functions that act on character tables (e.g. labels or namesystems).

They regard each row as a set element and perform simple set operations on the whole tables.

Two rows are considered equal if they only differ in trailing blanks.

Syntax	Meaning	Result
`unique` *tabA*	Extract unique rows	Char. table
tabA `index` *tabB*	Look up rows of *tabB* in *tabA*.	Row indices
tabA `isin` *tabB*	Flag rows of *tabA* if found in *tabB*	Boolean vector
tabA `with` *tabB*	Set union	Char. table
tabA `within` *tabB*	Set intersection	Char. table
tabA `without` *tabB*	Set difference	Char. table

6.5 Indented tables

Advanced This section, for programmers, documents the available support for indented labels.

These examples refer to a sample indented namesystem, called `PRODUCTS` :

```
All products
___Modems
______Internal
_________001
_________002
_________003
______External
_________004
_________005
_________006
_________007
___Hard disks
______009
```

- *Depths* ← `GETDEPTH` *Indented-table*

  Extract element depth from indented table.

  Depths will contain integers representing the depth of each line in the tree structure.

  For example, `GETDEPTH lablPRODUCTS` will produce

  ```
  0 1 2 3 3 3 2 3 3 3 3 1 2
  ```

- *Table* ← `GETLEAVES` *Indented-table*

  Extract atomic elements from indented table

  Table will contain the atomic rows of *Indented-table*, that is, the leaf elements in the tree structure.

  For example, `GETLEAVES lablPRODUCTS` will produce

  ```
  _________001
  _________002
  _________003
  _________004
  _________005
  _________006
  _________007
  ______009
  ```

- *Table* ← `UNDENT` *Indented-table*

  Remove indentation.

Table is a obtained from *Indented-table* removing the leading underscores and left justifying the labels.

For example, `UNDENT lablPRODUCTS` will produce

```
All products
Modems
Internal
001
002
003
External
004
005
006
007
Hard disks
009
```

6.6 Backend Modules

Advanced This section lists various modules, that are typically called

- in the **EXECUTE** command
- in direct APL programming.

The keyword `RCReport` that appears in the examples is a way to display the module return code.

- `doBALANCE` *TwoAxes*

  This backend program is used to compute the balance of inter-company accounts.

  Suppose you have a 3-way cube, with axes called "Accounts" "Companies-Debit" "Companies-Credit". You have the same population of Companies on axes 2 and 3, so that a single account may look as

Royalties	WHSmith	PanBooks	Springer	Addison
WHSmith	0	1100	900	0
PanBooks	400	0	15	100
Springer	170	180	0	1200
Addison	40	50	40	0

  After the call `doBALANCE 2 3` the same square table will be:

Royalties	WHSmith	PanBooks	Springer	Addison
WHSmith	0	700	730	-40
PanBooks	-700	0	-165	50
Springer	-730	165	0	1160
Addison	40	-50	-1160	0

In other words we compute the difference between a square table and its transpose.

See an example in procedure `INTER_COMPANY` in library `example2`.

- `doCALC` *axis formulae label*

  Advanced This backend program recomputes the current cube on dimension *axis* using the packet *formulae* (a 2-way char table) and resolving double-quote notations with *label* (a 2-way char table).

  For example, to run the database formulae on dimension 1 of the current cube, you would enter `doCALC 1 EXTI DI`

  TANGRAM's internal names of the current cube labels are `DI DA DB DC...` while the database formulae are kept in `EXTI EXTA EXTB EXTC...`

  If you use private packets of formulae, follow the rules of Admin formulae. If you use the double-quote notation, the variable *label* is used to translate this notation into equivalent indices. Otherwise *label* is unused.

  See an example in procedure `REDUNDANT` in library `example2`.

- `doDB2XML` *format filename*

  Export the current cube as an XML document.

  This utility writes the (real) current cube and associated labels to a file *filename* with extension `.xml`. The total width of the number format and the decimal places are controlled by *format*.

  The output can be displayed with a Web browser (and, of course, XML tools). The reverse transformation (from XML to TANGRAM) is also available for this XML document definition.

  For instance:

  ```
  RCReport  doDB2XML (12 2)   'htm/myfile.xml'
  ```

- `doINTERPOL` *count*

  This backend program will interpolate the current cube on axis 2, that is, it will insert *count* elements between existing periods and compute them as a straight-line interpolation.

  For example, to obtain monthly values from yearly data:

  ```
  doINTERPOL 11
  ```

Note that this operation is meaningful for stock-type variables (e.g. the accounts of a Balance Sheet) and not flux-type variables (e.g. the accounts of a Profit and Loss).

See an example in procedure `INTERPOLATE` in library `example2`.

- `doMODEL`

  3-way Create linear model.

  This module assumes that your current cube is in Standard Orientation (Variables, Periods, Items) and uses linear algebra to find the best linear combination of $N-1$ Variables to estimate the *last* Variable.

  The secondary cube initially must not exist and is assigned the model coefficients. The process concerns Variables and Items and is iterated on Periods.

  After execution of `doMODEL` you can recompute the (estimated) last variable by calling **COMPUTE** and providing a formula with the keyword `useMODEL`.

  See procedure `MODEL` in library `example3`.

- `doRANK`

  3-way Ranking en masse.

  This module computes rank positions (1 for the highest value, 2 for the second-highest etc.).

  The elements to be ranked are always the *items* (whatever you have on dimension 3). Every *variable-period* pair (whatever you have on dimensions 1 and 2) produces an independent ranking.

  For example `doRANK` will, say, assign the value 5 to a cell associated with these labels

  ```
  Variable costs/revenues
  Fiscal Year 2007
  Alpha Omega Ltd
  ```

  if in 2007 Alpha Omega Ltd happens to have the fifth highest ratio among all companies in the current cube.

  Missing values appear last in every ranking.

  The cube acquires a new dimension, with elements `Absolute` and `Rank`.

- `doTOXML` *format filename*

  Export the current cube as an XML document.

  This utility writes the (real) current cube and associated labels to a file *filename* with extension `.xml`. The total width of the number format and the decimal places are controlled by *format*.

The output can be displayed with a Web browser (and, of course, XML tools).

For instance:

```
RCReport  doTOXML (10 0)   'htm/test.xml'
```

6.7 Miscellaneous utilities

Advanced This section lists various keywords and utilities, that are typically used in the Questions and Answers Dialog Box to provide the result that an input box is expecting.

As usual, they can also be used in direct APL programming and in the **EXECUTE** module.

- *table-name* `COUNT` *dimensions*

  This keyword can be handy to express a general answer that depends on some element count in a cube or in a namesystem.

  For example, in the creation of a new database, the correct size might depend on the number of elements in some namesystem, like Customers or Products.

  This allows for three slightly different calls:

  - If the left argument is missing, the result refers to the current cube (real or virtual).

    For ex. `COUNT 2 3` may return `21 40`

    If the right argument is empty, all dimensions are returned: For ex. `COUNT ''` may return `5 21 40 11`
  - If the left argument is a variable of any kind, the result returns its dimensions.

    For ex. `xT  COUNT 1 2 3` may return `3 6 12`
  - If the left argument is the name of a namesystem, the result is its row- or column- count.

    For ex. `'Provinces' COUNT 1` may return `26`

- `DIGEST` *n*

  Cube signature, cube comparison.

  If you need to compare two (real, current) cubes (to see that two solutions produced the same cube) you may issue `DIGEST` on each of them and compare the results.

  `DIGEST` produces two n-digit values; the first encodes all cube labels and dimension names, the second the cells. For example:

```
      DIGEST  6
324122 871771
```

`DIGEST` is sensitive to trailing blanks, element and cell permutation and to 3 significant digits in cell values.

This means that if a cell value is changed from 0.004512 to 0.004513 the signature remains the same.

The number of significant digits defaults to 3 but can be coded as the left argument of `DIGEST`. For example, to test 5 significant digits:

```
5 DIGEST 6
```

Both arguments have a maximum of 9.

Of course, two results can only be compared for equality and do not measure any closeness.

If a second cube produces `625330 871771` you can only conclude that the cells are the same and the label structure differs.

- *A* `FORCE` *B*

  This module replaces a value with another (in all cells of the current cube).

  The argument *A* is the value to be replaced in the current cube, while *B* is the replacement value.

  For example:

  ```
  ND FORCE 0
  ```

  TANGRAM handles missing values `ND` as different from zeroes. In some cases you may safely assume that missing values are to be treated as zeroes. Or you may want to open a database from Excel, making sure that missing values are replaced with a chosen value.

- *thresholds* `histo` *num-table*

  Histogram of values.

  `histo` applies to any numerical table and counts the cells that fall into each interval. *thresholds* is a vector of N increasing values and the result is a vector of $N + 1$ counts. Cells equal to a threshold value are counted in the left interval.

  For instance:

  ```
        0 10 100 1000 histo T
  0 0 25 120 2100
  ```

- *style* `INFO` *char-table*

  This is the standard way to display text.

  The variable *style* influences the type of output:

  0 Information

  1 Warning

2 Error

3 Trapped program failure

4 Printout page

5 User comment

If you wish to hand-insert comments in procedures, use style 5. To do this, you simply edit a procedure and insert a line like

```
5 INFO 'We now proceed to update the database.'
```

- `LABEL` *dimension*

  Labels of the current cube

  This keyword returns the labels of the current cube on a dimension. The result is a two-way character table. Note that the keyword can be used to read, not to assign the labels.

  Example 1

  To capture the labels of dim. 3 in a variable, you may use **EXECUTE** and enter the formula `userPRODUCTS : LABEL 3`

  Example 2

  To highlight totals in red during a **BROWSE** session, you may enter the formula

  ```
  red IF 1 INLIST (LABEL 1) SPOT 'TOTAL'
  ```

  In fact `(LABEL 1) SPOT 'TOTAL'` returns the line indices where the word 'TOTAL' appears and a list of indices is a legitimate input to `INLIST` .

  Example 3

  In the module **GLOBAL** you wish to multiply T by 1.33 only where the labels of the third dimension contain 'Gmbh':

  ```
  ((1.33 TIMES T) IF 3 INLIST (LABEL 3) SPOT 'Gmbh') OrElse T
  ```

- `Sw`

  Stop watch.

  This keyword returns the number of seconds elapsed from the previous call (or from the beginning of the worksession). The result is numeric.

  For example, simply type `Sw` in the **EXECUTE** input window.

- `TS`

  Time stamp.

  This keyword returns date and time (from the system clock) in this character format

  `DD/MM/YYYY HH:MM`

  Simply type `TS` in the **EXECUTE** input window.

  The output may look like:

  ```
  31/12/2007 24:55
  ```

- `xLABEL` *dimension*

  Labels of the secondary cube.

  This keyword returns the labels of the secondary cube on a dimension. The result is a two-way character table. Note that the keyword can be used to read, not to assign the labels.

  Example

  To compare the labels on dim. 2 of current and secondary cube, extracting those that appear only in the secondary cube, you may use **EXECUTE** and enter the formula

  ```
  userEXCEPTIONS : (xLABEL 2) without LABEL 2
  ```

6.8 The APL interpreter

Advanced Strictly for programmers.

If you have some knowledge of the APL language (and a development version of the Dyalog interpreter) you can add your own code or you can formulate your replies in APL.

When you insert APL code in procedures or use the APL session (in direct execution), remember that you are on your own, meaning that several TANGRAM services are not available:

- The exclamation mark `!` is not translated into the APL comment.
- The apostrophe is not translated into the APL quote.
- The minus sign next to a number is not translated into the APL high-minus.
- The colon `:` is not translated into the assignment arrow.
- You get APL error messages, which are not translated. The error trap restarts TANGRAM, unless you reassign it.

- The APL names of modules, procedures and namesystems have prefixes `go` , `proc` and `labl` . A namesystem, called `PRODUCTS` in a TANGRAM box, in APL is `lablPRODUCTS`
- Similarly, TANGRAM facilitates coding by knowing the context of a question. If you compute on axis 4 of a 5-way table, an element can be denoted as `[10]` . In programming you must use all indices, like `T[;;; 10 ;]`
- The chosen axis is also known to TANGRAM, not to APL. For example, you may use `MYTABLE SPOT 'budget'` and not simply `SPOT 'budget'` For example, `T[; EVERY 5;]` is not understood.
- The double-quote notation (like `"Net profit"`) is not supported.

♢ A programmer finds it useful to know the internal names of TANGRAM variables (global variables to be read, not assigned):

- The dimension names are in `dd`
- The current cube is `T` (when real).
- The labels are `DI DA DB DC ...`
- The Admin formulae are `EXTI EXTA EXTB EXTC ...`
- The indices of read-only elements are `XI XA XB XC ...`
- The secondary cube dimension names are in `xdd`
- The secondary cube is `xT`
- The secondary cube labels are `xDI xDA xDB xDC ...`
- The current database is in `fileH`
- The current library is in `fileL`
- Printed outputs are appended to `fileP`

♢ A few reminders on APL.

The APL keyboard is obtained with Ctrl N. The text keyboard is obtained with Ctrl O.

APL names are case-sensitive. Execution is right to left, unless you use parentheses.

Both APL primitives and user-defined functions accept a right argument and (optionally) a left argument. APL primitives work with two arrays of the same shape (`T + xT`) or an array and a singleton (`T + 1000`).

The syntax along a line (and within a parethesis) simply states that the right argument of a function is the result of the expression on its right.

Consider, for instance, the line:

```
1000 ROUND +/ T DIVIDE xT + 1
```

The four functions are called in the order: `+` , then `DIVIDE` , then `+/` , then `ROUND`

They all have two arguments, except `+/`

The right argument of `ROUND` is the result of `+/ T DIVIDE xT + 1`

If the result of the line is not assigned to a variable, as in this instance, it is displayed on the (APL session) screen.

Finally, make sure you know the basic commands `)SAVE` and `)LOAD` to save the workspace with your work.

6.9 Operators

Advanced Advanced

From an APL programmer's point of view, so far we have encountered only two types of objects: variables (like `T`) and functions (like `PLUS`).

Operators are objects of a different nature that modify the way a function works. For example, `+/[3] T` reduces the current cube `T` along the 3rd axis, obtaining a result of lower rank (a table with the same axes except the 3rd).

In this case `+` is the function, `/` the operator and `[3]` the axis selector.

Any other function can replace the `+` . For example, `PLUS /[3] T` will produce a similar result handling missing values and `×/[3] T` will multiply the cells along the 3rd axis.

The axis selector defaults to the last axis of the table. The formula `+/,T` will produce the sum of all cells of a table `T` of any rank.

A different operator `\` computes the running or progressive result of a function, so that `+\[3] T` is a table of the shape of `T` with running totals on the 3rd axis. The effect is similar to that of the module **TODATE** but `\` works on any axis, on any table and with any function.

If you have a 3-way cube with dimension names Variables, Years, Customers, you may want to compute the running total of a variable on all Customers. This can be done via **EXECUTE** and a formula like `T[1;12; ] : +\ T[1;12; ]`

The axis selector is omitted, because the indexing on Variable 1 and Year 12 reduces the table to a single string. Omitting the indices of Customers, you intend all existing elements on that axis.

There are six native operators in Dyalog APL, but only the two we have seen are occasionally useful to an OLAP user.

As it is the case with functions, *native* operators can be augmented with *defined* ones and TANGRAM defines two that apply to hypercube processing.

- *LVar Function* `sparse` *RVar*

  A few defined functions (like `PLUS MINUS TIMES DIVIDE`) modify the native arithmetic operations to handle missing values.

The operator `sparse` allows you to obtain a similar behaviour for any other APL function.

Examples

To modify the power function `*` you may write a **GLOBAL** formula `T * sparse 2` to obtain the square of each non-missing cell.

The Boolean function `=` is not aware of missing values. You can modify it as in, for example, `T = sparse xT` to yield a three-valued result (in the range $0, 1, ND$).

- *Axes Function* `along` *Var*

  The operator `along` reduces a table with a function (similarly to `+/ T`) and then expands the result to be of the same shape as the original table (to ease further processing between tables of the same shape). Its left argument is a list of axes, followed by the function; its right argument is an array.

  Examples

  `2 + along T` will compute the totals on axis 2 and then replicate the result to reobtain the shape of `T`. All elements on axis 2 will have the same contents (the total).

  `(2 3 + along T) ÷×/COUNT 2 3` will obtain the average on axes 2 and 3.

  The *Function* can in turn be modified by an operator, as in

  ```
  1 ( OR sparse ) along T < sparse 0
  ```

Appendix A

Keyword index

Appendix B

How-to index

- **Analyze data from several cubes**

  The basic tool is the secondary cube (page 84).

  Modules **DOCK** (page 105), **MERGE** (page 59) and **TRANSFER** (page 97) are relevant, too.

- **Create a database**

  NEWDB (page 91) and **CROSSDB** (page 92) create empty databases, **BUILDDB** (page 100) imports a flat file. **SAVEAS** (page 64) writes the current cube to file.

- **Compare two cubes**

  If two cubes have the same labels, see **DELTA** (page 55) and **BROWSE** (page 26) with a color formula.

  If they have unrelated elements, see **MERGE** (page 59).

  If you need a yes/no answer for the equality of whole cubes, see `DIGEST` (page 137).

- **Compute the cells**

  General-purpose tools for computing:

  - **COMPUTE** (page 35) for a new element
  - **GLOBAL** (page 47) to recompute all cells
  - **EXECUTE** (page 22) for single cells
  - **RUNFORMULAE** (page 111) executes formulae on the whole database.

  Special-purpose tools: `doBALANCE` and `doCALC` (page 134).

- **Cost allocation**

  Use either **BLOWUP** (page 56) or **DISAGGREGATE** (page 57).

- **Currency conversion**

  The simplest way is to have the cells to be converted in the current cube and the rate in the secondary cube and use **GLOBAL** (page 47) with a formula `T TIMES xT` .

  But there are variations on the theme. At times you need to expand the cube containing the conversion rates: see **PROMOTE** (page 62) and **SELECT** (page 20) with element repetition.

  At other times a **GLOBAL** formula (page 47) with the scope limited by the keyword `SUB` solves the problem.

- **Data import-export**

  See the diagram at the end of the manual for a synopsis. Emphasis is on the exchange of XML, flat files and other formats that can be imported or exported in one-shot (as opposed to the cooperation of many complex software components).

  TANGRAM cubes can be directly read from Excel or transformed into Excel worksheets.

  See also **BROWSE** (page 26) for copy-and-paste of cube pages with Windows applications.

- **Delta analysis**

  For instance, how to find out if we are below budget because of lower sales, lower prices, higher discounts etc.

  The main tool for this is **CONTRIBUTE** (page 49).

- **Demo**

  Choose library `example` in the Main Menu, then select Start worksession and Run a procedure . Leave the "Procedure to run" field empty to see the available procedures.

  These are short (and verbose) procedures that show the use of a few modules at a time.

  During step-by-step procedure execution, leave the default answers and press Enter to proceed (or click "OK" to leave the grid).

  Libraries `example2` and `example3` contain more advanced procedures (some of these create new databases and require a full installation of TANGRAM).

- **Drill down**

  For instance, how to move from Product-families to Products or from Years to Months.

  Use **ROLLUP** (page 43) on fewer facets, or, better, use **GROWTREE** (page 45) to see all levels at a glance.

- **Expand summary data with more detail**

  For example, expand a concise budget to the full size of actual data, or allocate global costs with a *driver*.

  Use either **BLOWUP** (page 56) or **DISAGGREGATE** (page 57).

- **Logical view**

  How to change the logical view of a cube, say, seeing it by Customer and Product and then by Month and Salesman.

  BROWSE and the programs that produce printouts let you select the axes on rows and columns.

  TRANSPOSE (page 52) and **SHUFFLE** (page 63) rotate the cube in memory.

  CROSSOVER (page 53) restructures the cube in a more powerful way, moving single facets across dimensions.

  LABELJOIN (page 25) brings in new facets from a namesystem (for instance, appending Customer-class and Customer-size to Customer codes).

- **Missing values** (*alias*: Nulls)

  See **MISSING** (page 34) for a map of missing values. See keywords like `PLUS` (page 35) for computing with propagation of missing values. Use **BROWSE** (page 26) with the color formula `BLACK IF T=ND` to dim missing cells.

  Use **MERGE** (page 59) to merge two cubes with scattered missing values. See also the keywords `IF ELSE OrElse` (page 40) and `FORCE` (page 137).

- **Periodicity of time axis**

  In TANGRAM you do not declare whether your time periods represent months, quarters, years,... or are heterogeneous.

  To reduce the periodicity (say, from months to quarters) you use a different technique according to the type of variable.

  For a variable "Current assets" (end of period) you would simply **SELECT** your periods with `EVERY 3`

  For a variable "Sales" (in the period) you would use **TODATE, SELECT** with `EVERY 3` then **TOPERIOD**.

  See `doINTERPOL` (page 134) to increase the periodicity with a straight-line interpolation.

  Other relevant modules: **MERGE BLOWUP DISAGGREGATE MOVINGAVERAGE**.

- **Populate a database**

See **DATAENTRY** (page 95), **BUILDDB** (page 100), **LOADDB** (page 96) and **DOCK** (page 105).

RANDOMDB (page 104) and **FILLDB** (page 103) produce test values.

- **Print a cube**
  - Save it, open it from Excel and use styling templates or Excel features.
  - Use **REPORT** (page 73), **MASKPRINT** (page 74) or **DRAFTPRINT** (page 77).
  - Export it with **TOLATEX** (page 71) or **TOHTM** (page 69).
  - Export it with `doDB2XML` or `doTOXML` (page 134) and use XML tools.
- **Rank of an element**

  To obtain element rank from cell values, see `doRANK` (page 136).

  To sort on the values, see **SELECT** (page 20) and keyword `BYVALUE`.

  To select the largest elements that make up x percent of the total, refer to `MAJOR` (page 127).
- **Roll up on a cube axis**

  See **ROLLUP** (page 43) if you have facets that define the aggregation, see **GROWTREE** (page 45) if you hand-define an aggregation tree with variable depth.
- **Resize a cube**

  The current cube in memory changes its size (or rank) as an effect of your commands and can be saved by **SAVEAS** (page 64).

  A database size can be redefined by the modules **RESIZEDB** (page 93) and **MOVEDB** (page 94).
- **Run TANGRAM in batch mode**

  See unattended execution of procedures (page 15).
- **Seasonal behavior**

  Use **BLOWUP** (page 56) to expand a single period with the seasonal behavior of the seconday cube. Use **COMPUTE** (page 35) to de-seasonalize element 1 with the seasonal behavior of element 2, for example: `[1] DIVIDE [2]`
- **Sparsity and performance**

  When designing very large databases (say, above 50 million cells) make sure you read the relevant chapter (page 87), then estimate the size, call **NEWDB** (page 91) and read its Database Design report.

  You then create the database and measure the performance, with special attention to the time-consuming operations (**RUNFORMULAE, BUILDDB, LOADDB**).

  BUILDDB (page 100) will also assist you to avoid excessive sparsity.

- **Sort the data in a cube**

  Use **SELECT** (page 20) on a dimension at a time. To sort on the labels, use the keywords `AZ` or `BYLABEL`. To sort on the cells, use the keyword `BYVALUE`.

  To sort each printout page differently, see also **DRAFTPRINT** (page 77).

- **Spot abnormally high and low values**

  The main tool is **BROWSE** (page 26) and its color formula.

  See also **QUERY** (page 30) and **GROWTH** (page 79).

  You can also use **FILLDB** (page 103) to spot cells that contribute more or less than their share to the total. See procedure `FILLDB_SPOT` in library `EXAMPLE2` for details.

- **Summary of large populations**

  If you wish to sample the population, select using keywords like `EVERY 10` or `10 ? 100` (to extract 10 different random elements out of 100).

  If you wish statistical summary measures, use **COMPUTE** (page 35) with either `STAT` or `PERCENTILE`.

  If you wish to group the elements on a measure, see **RANGE** (page 34). If you need to ad-lib questions on the population, see **QUERY** (page 30).

- **Time axis**

  All time-related operations are grouped in Section 3.7 (page 78). They all require a 3-way cube with time periods on axis 2.

  More sophisticated models for smoothing and extrapolation can be expressed with **COMPUTE** (page 35).

  See also Time-Shift keywords (page 42), which apply to N-way cubes.

- **Undo an operation**

  You can save the cube before a critical command with **SAVEAS** (page 64) or save the worksession and redo all commands except the last.

Appendix C

File extensions

Extension	Description	Default directory
`.hlm`	Database file (cells)	`hlm\`
`.sf`	Database file (structure)	`hlm\`
`.hld`	Database lock file	`hlm\`
`.lib`	Library file	`lib\`
`.log`	Editable log file	`log\`
`.txt`	Editable text file	`txt\`
`.xcp`	Rejected records (**LOADDB**)	`txt\`
`.htm`	HTML output file (**TOHTM**)	`htm\`
`.tmp`	HTML template (**TOHTM**)	`htm\`
`.xml`	XML file	`htm\`
`.msk`	Mask file (**MASKPRINT**)	`msk\`
`.tex`	Editable LaTeX file (**TOLATEX**)	`tex\`

Appendix D

Bibliography on DSS OLAP

The software family to which 3-way TANGRAM belongs is rapidly evolving.

To be informed of the latest technology, see the *newsgroup* `comp.databases.olap`

A good bibliography on OLAP, DSS, EIS is maintained by Prof. Daniel Power and Prof. Paul Gray and is reproduced here with their kind permission, to provide an historical view of the foundations.

The up-to-date bibliography can be found at the URL
`dssresources.com`

D.1 Books by author

- Adelman, L. Evaluating Decision Support and Expert Systems. New York, NY : John Wiley & Sons, 1992. ISBN 0-471-54801-4. [text] [This is a textbook that deals with both DSS and ES topics]
- Alter, S. Decision Support Systems : Current Practice and Continuing Challenges. Reading, Mass. : Addison-Wesley Pub., 1980. 316 p. ISBN 0-201-00193-4. [classic] [text] [This is a DSS classic with a strong emphasis on practice in the late 1970s]
- Andriole, S. J. Handbook of Decision Support Systems. (1st ed.) Blue Ridge Summit, PA : TAB Professional and Reference Books, 1989. 248 p. [proc]
- Baecker, R. M.. (editor) Readings in Groupware and Computer-Supported Cooperative Work: Assisting Human-Human Collaboration. San Mateo, CA: Morgan Kaufman Publishers, 1993. 882 p. [team] [book review]
- Bennett, J. L. (editor) Building Decision Support Systems . Reading, MA: Addison Wesley , 1981 ISBN 0-201-00563-8. Table of Contents [classic] [text] [This is a classic set of readings that focus on DSS design]
- Bigdoli, H. Decision Support Systems: Principles and Practices. St.Paul, MN: West Publishing, 1989. 0-314-46560-X. [text]
- Bird, J. Executive Information Systems Management Handbook. Blackwell, 1991. 141 p. [eis] [book review].
- Boulden, J. B. Computer-Assisted Planning Systems. New York: McGraw-Hill Book Company, 1975. 277 p. ISBN 0-07-006657-4. [text]
- Bonczek, R. H., C. W. Holsapple, and A. Whinston. Foundations of Decision Support Systems. Academic Press, 1981 ISBN 0-12-113050-9 [classic] [text]. [Classic DSS textbook that stressed a modeling/management science approach to DSS]

- Bostrom, R. P., R. T. Watson, and S. T. Kinney (editors). Computer Augmented Teamwork: A Guided Tour. New York: Van Nostrand Reinhold, 1992. ISBN 0-442-00277-7. [gdss] [book review]
- Brophy, P. Management Information and Decision Support Systems in Libraries. Aldershot, Hants, England ; Brookfield, Vt., USA : Gower, 1986. 158 p. [app].
- Bui, T. X. Co-Op: A Group Decision Support System for Cooperative Multiple Criteria Group Decision Making. Berlin: Springer Verlag, 1987. ISBN 0-387-18753-7 [gdss] (Vol. 290 in the Lecture Notes in Computer Science series published by Springer Verlag)
- Burkan, W. C. Executive Information Systems: From Proposal through Implementation. New York : Van Nostrand Reinhold, 1991. 174 p. [eis] [book review]
- Carter, G. M., M. Murray, R. G. Walker, and W. E. Walker Building Organizational Decision Support Systems. Boston, MA: Academic Press, 1992. 358 p. ISBN 0-12-732070-9 [text]
- Clowes, K. W. The Impact of Computers on Managers. Ann Arbor, Mich. : UMI Research Press, 1982. 190 p. ; Bibliography: p. 179-181.
- Coleman, D. and R. Khanna (editors). Groupware: Technology and Applications. Englewood Cliffs, NJ: Prentice-Hall PTR, 1995. 576 p. ISBN 0-13-305194-3 Table of Contents [gdss] [book review]
- Fink, C. Knowledge-based Systems for Financial Executives. Morristown, N.J. : Financial Executives Research Foundation, 1991.63 p. "Deloitte & Touche Information Technology Research." [app]
- Ginzberg, M. J., W. Reitman, E. A. Stohr (editors). Decision Support Systems. Amsterdam: North Holland, 1982. [proc].
- Gonzalez, A. J. and D. D. Dankel. The Engineering of Knowledge-based Systems: Theory and Practice. Englewood Cliffs, NJ: Prentice-Hall, 1993. ISBN 0-13-276940-9. Table of Contents [es]
- Gray, P. (editor). Decision Support and Executive Information Systems. Englewood Cliffs, N.J.: Prentice-Hall, Inc.: 1994, ISBN: 0-13-235789-5 [proc]
- Gray, P. Visual IFPS/Plus for Business Englewood Cliffs, N.J.: Prentice-Hall, Inc.: 1996. ISBN: 0-13-185604-9 Table of Contents [text]
- Heymann, H. G. and R. Bloom. Decision Support Systems in Finance and Accounting. New York : Quorum Books, 1988. 195 p. [app].
- Holsapple, C. W. and A. B. Whinston. Decision Support Systems: A Knowledge Based Approach. Minneapolis, MN.: West Publishing, Inc., 1996. Book website link. [text]
- Hopple, G. W. The State-of-the-art in Decision Support Systems Wellesley, Mass. : QED Information Sciences, 1988. 246 p. [proc]
- House, W. C. (editor). Decision Support Systems: A Data-Based, Model-Oriented, User-Developed Discipline. Petrocelli, 1983. ISBN 0-89433-225-2 [proc].
- Humphreys, P., L. Bannon, A. McCosh, P. Migliarese, J. Pomerol (editors). Implementing Systems for Supporting Management Decisions: Concepts, Methods and Experiences. London, UK: Chapman & Hall, 1996, 379 p.; ISBN 0-412-75540-8. [proc]
- Humphreys, P., O. Svenson and A. Vari (editors). Analysing and Aiding Decision Processes . Budapest, Hungary: Akademiai Kiado, 1983, 543 p.; ISBN 963-05-32891. [proc]
- Inmon, W. H. Building the Data Warehouse. New York: John Wiley & Sons, 1993. ISBN 0-471-56960-7. [dw] [book review]
- Inmon, W. H. and R.D. Hackathorn Using the Data Warehouse. New York: John Wiley & Sons, 1994. ISBN 0-471-05966-8. [dw] [book review]
- Jelassi, T., M. R. Klein, and W. M. Mayon-White (editors). Decision Support Systems: Experiences and Expectations. Amsterdam, NL: North-Holland, 1992. ISBN 0-444-89673-2. [proc]

- Jessup, L. M. and J. S. Valacich (editors). Group Support Systems : New Perspectives. New York : Macmillan ; Toronto : Maxwell Macmillan Canada, 1993. ISBN: 0-02-360625-8. [gdss] [book review]
- Johansen, R. Groupware: Computer Support for Business Teams. New York: Free Press, 1988. ISBN 0-02-916491-5 [team] [book review]
- Johansen, R., Sibbet, D., Benson, S., Martin, A., Mittman, R. and P. Saffo. Leading Business Teams: How Teams Can Use Technology and Group Process Tools to Enhance Performance. Reading, MA: Addison Wesley, 1991. 0-201-52829-0. [team] [book review]
- Keen, P. G. W. and M. S. Scott Morton. Decision Support Systems: An Organizational Perspective. Reading, MA: Addison-Wesley, Inc., 1978 ISBN 0-201-03667-3. Table of Contents [classic] [text].
- Kelly, S. Data Warehouse: The Route to Mass Customisation. Chichester, UK: John Wiley & Sons.,1995 ISBN 0-471-95082-3 [dw].
- Kimball, R. The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses. New York: John Wiley & Sons, 1996. ISBN 0-471-15337-0. (includes CD-ROM) [dw] [book review]
- Klein, M. R. and L. B. Methlie. Knowledge-based Decision Support Systems with Application in Business. (2nd ed.). New York, NY: John Wiley & Sons, 1995 ISBN 00471-95295-8 [es] [text].
- Leigh, W. E. and M. E. Doherty. Decision Support and Expert Systems. Cincinnati, OH: South-Western Publishing Company, 1986. 454 p. ISBN 0-538-10910-6 [text].
- Lotfi, V. and C. Pegels. Decision Support Systems for Operations Management and Management Science. (3rd. ed.) Homewood, IL : Irwin, 1996. 480 p. ISBN 0-256-11559-1 (includes disk) [app]
- Mallach, E. G. Understanding Decision Suppot and Expert Systems. Burr Ridge, IL: Richard D. Irwin, Inc., 1994. ISBN 0-256-11896-5.
- Mason, R. O. and E. B. Swanson. Measurement for Management Decision. Reading, MA; Addison Wesley Publishing, 1981. ISBN 0-201-04646-6.
- McLean, E. R. and H. G. Sol (editors). Decision Support Systems: A Decade in Perspective Amsteram, NL: North-Holland, 1986. 251 p. ISBN 0-444-70037-4 Table of Contents [proc](Note: Proceedings of IFIP Working Group 8.3, Netherlands, 1986)
- McNichols, C. W. and T. D. Clark. Microcomputer-Based Information and Decision Support Systems for Small Businesses: A Guide to Design and Implementation. Reston, VA: Reston Publishing (A Division of Prentice-Hall), 1983. ISBN 0-8359-435 [app].
- Methlie, L. B. and R. H. Sprague (editors). Knowledge Representation for Decision Support Systems . Amsterdam, NL: North-Holland, 1985, 267 p.; ISBN 0-444-87739-8. Table of Contents [proc]
- Mockler, R. J. Knowledge-based Systems for Management Decisions. Englewood Cliffs, N.J.: Prentice-Hall, Inc.: 1989. ISBN 0-13-516907-0. Summary [es] [text]
- Mockler, R. J. Knowledge-based Systems for Strategic Planning. Englewood Cliffs, N.J.: Prentice-Hall, Inc.: 1989. ISBN 0-13-516915-1 [es] [text].
- Mockler, R. J. Computer Software to Support Strategic Management Decision Making. New York: Macmillan, 1992. ISBN 0-02-381895-6 [text].
- Nagel, S. S. (editor). Computer-aided Decision Analysis : Theory and Applications. Westport, Conn. : Quorum Books, 1993, 284 p [proc].
- Olson, D. L. and J. F. Courtney, Jr. Decision Support Models and Expert Systems. New York: Macmillan, 1992. ISBN 0-02-389340-0 [text].
- Opper, S. and H. Fersko-Weiss Technology for Teams: Enhancing Productivity in Networked Organizations. New York : Van Nostrand Reinhold, 1992, 181p. ISBN 0-44223928 [team] [book review]

- Paller, A. with R. Laska. The EIS Book : Information Systems for Top Managers Homewood, IL : Dow Jones-Irwin, 1990, 217 p. [eis] [book review]
- Poe, V. with contributions by L. L. Reeves. Building a Data Warehouse for Decision Support . Upper Saddle River, NJ: Prentice Hall PTR, 1996, 202 p. ISBN 0-13-371121-8. Table of Contents [dw] [book review]
- Rockart, J. F. and C. V. Bullen (editors). The Rise of Managerial Computing: The Best of The Center for Information Systems Research. Homewood, Ill: Dow Jones-Irwin, 1986. ISBN 0-87094-757-5. [book review]
- Rockart, J. W. and D. W. DeLong. Executive Support Systems: The Emergence of Top Management . Homewood, IL.: Dow Jones Irwin, 1988. 280 p. [eis] [book review]
- Sage, A. P. Decision Support Systems Engineering New York: John Wiley, 1991. ISBN 0-471-53000-x. [text]
- Sauter, V. Decision Support Systems. New York, NY: John Wiley & Sons, 1997. ISBN 0-471-31134-0. Web link [text].
- Scott Morton, M. S. Management Decision Systems: Computer-based Support for Decision Making. Boston, MA: Division of Research, Graduate School of Business Administration, Harvard University, 1971. 216 p. ISBN 0-87584-090-6 [classic].
- Shakun, M. F. Evolutionary System Design: Policy Making Under Complexity and Group Decision Support Systems. Oakland, CA: Holden-Day Inc., 1988. 287 pp. ISBN 0-8162-7819-9
- Silver, M. S. Systems that Support Decision Makers: Description and Analysis. Chichester: John Wiley & Sons, 1991 [text].
- Sol, H. G. (editor). Processes and Tools for Decision Support. Amsterdam: North Holland, 1983. Table of Contents [proc]
- Snoyer, R. S. and G. A. Fischer (editors). Everyone's Support Systems : A Complete Guide to Effective Decision Making using Microcomputers. Homewood, Ill. : Business One Irwin (Chantico Publishing Company, Inc.), 1993, 405 p. [app]
- Sprague, R. H. and E. D. Carlson. Building Effective Decision Support Systems Englewood Clifts, N.J.: Prentice-Hall, Inc.: 1982, ISBN: 0-13-086215-0 Table of Contents [classic] [text]
- Sprague, R. H. and H. J. Watson (editors). Decision Support Systems : Putting Theory into Practice. 3rd ed. Englewood Clifts, N.J. : Prentice Hall, 1993. 437 p. ISBN 0-13-036229-8 [readings] [text].
- Sprague, R. H. and Hugh J. Watson (editors). Decision Support for Management Englewood Clifts, N.J.: Prentice-Hall, Inc.: 1996, ISBN: 0-13-396268-7 Table of Contents [readings] [text]
- Stohr, E. A. and B. R. Konsynski (editors). Information Systems and Decision Processes Los Alamitos, CA: IEEE Computer Society Press, 1992. 349 p. ISBN 0-8186-2802-2. [proc] (Note: TIMS College on Information Systems and NSF sponsored study of the future of decision systems)
- Thierauf, R. J. Decision Support for Effective Planning and Control: A Case Study Approach. Engelwood Cliffs, N.J: Prentice Hall, 1982. ISBN 0-13-198234-6 [text]
- Thierauf, R. J. User-Oriented Decision Support Systems: Accent on Problem Finding. Englewood Cliffs, NJ: Prentice Hall, 1988. ISBN 0-130940412-0. [text]
- Thierauf, R. J. Group Decision Support Systems for Effective Decision Making: A Guide for MIS Practitioners and End Users. New York : Quorum Books, 1989 [gdss].
- Thierauf, R. J. Executive Information Systems: A Guide for Senior Management and MIS Professionals. New York : Quorum Books, 1991. xix, 364 p. [eis].
- Trippi, R. R. and E. Turban. Investment Management: Decision Support and Expert Systems. Danvers, MA: boyd & fraser, 1990 0-87835-451-4 [app].

- Turban, E. Decision Support and Expert Systems: Management Support Systems. (4th edition) Englewood Cliffs, N.J.: Prentice-Hall, Inc.: 1995. ISBN: 0-02-421701-8. Table of Contents [text]
- Watson, H. J., R. K. Rainer, and G. Houdeshel. Executive Information Systems: Emergence, Development, Impact. New York, NY: John Wiley & Sons, 1992. ISBN 0-471-55554-1 [eis] [book review]
- Young, L. F. Decision Support and Idea Processing Systems. Dubuque, IA: Wm. C. Brown, 1989. ISBN 0-697-00742-01 [text].

D.2 Books by topic

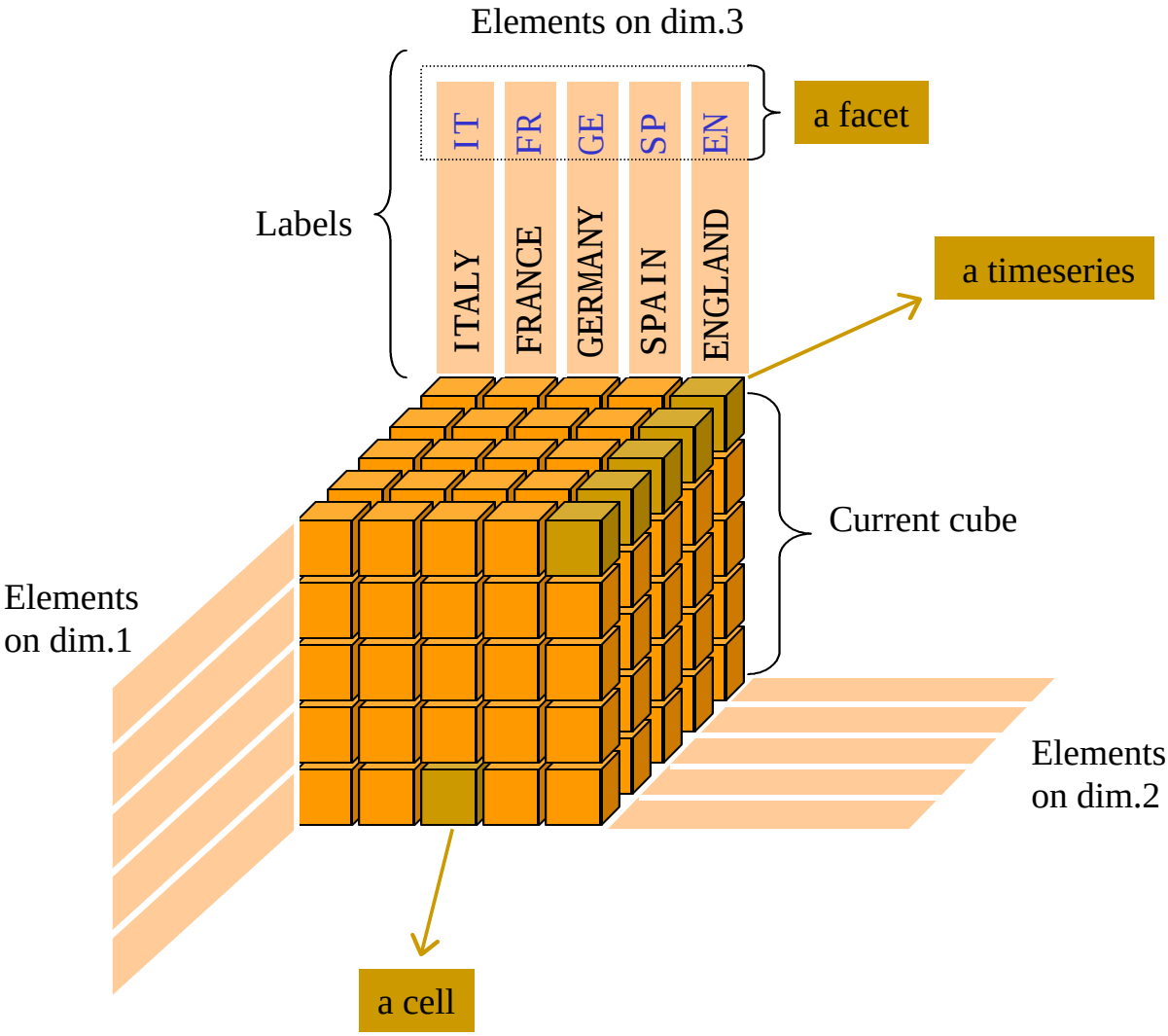
- Data Warehousing
  - Inmon, 1993
  - Inmon and Hackathorn, 1994
  - Kelly, 1995
  - Kimball, 1996
  - Poe, 1996
- DSS Applications
  - Brophy, 1986
  - Fink, 1991
  - Heymann and Bloom, 1988
  - Lofti and Pegels, 1991
  - McNichols and Clark, 1983
  - Snoyer and Fischer, 1993
  - Trippi and Turban, 1990
- DSS Classics
  - Alter, 1980
  - Bennett, 1981
  - Bonczek et al., 1981
  - Keen and Scott Morton, 1978
  - Scott Morton, 1971
  - Sprague and Carlson, 1982
- DSS Proceedings/Readings
  - Andriole, 1982
  - Ginzberg et al., 1982
  - Gray, 1994
  - Hopple, 1988
  - House, 1983
  - Humphreys et al., 1996
  - Humphreys et al., 1983
  - Jelassi et al., 1992
  - McLean et al., 1986

  - Methlie and Sprague, 1985
  - Nagel, 1993
  - Sol, 1983
  - Sprague and Watson, 1993
  - Sprague and Watson, 1996
  - Stohr and Konsynski, 1992

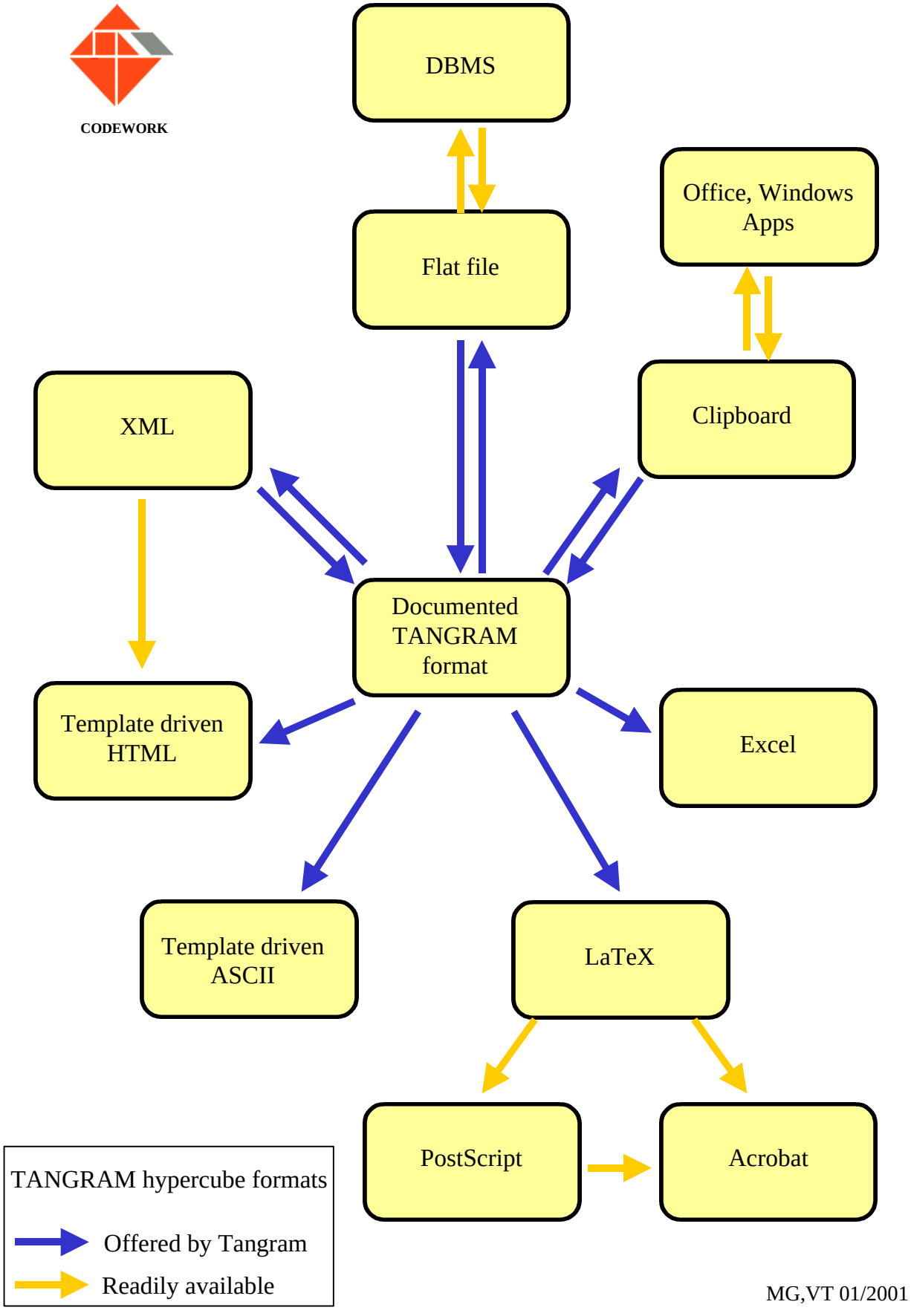
- DSS Text Books
  - Adelman, 1992
  - Alter, 1980
  - Bennett, 1981
  - Bigdoli, 1989
  - Boulden, 1975
  - Bonczek et al., 1981
  - Carter et al., 1992
  - Gray, 1996
  - Holsapple and Whinston, 1996
  - Keen and Scott Morton, 1978
  - Klein and Methlie, 1995
  - Leigh and Doherty, 1986
  - Mallach, 1994
  - Mockler, 1989a
  - Mockler, 1989b
  - Mockler, 1992
  - Olson and Courtney, 1992
  - Sage, 1991
  - Sauter, 1997
  - Silver, 1991
  - Sprague and Carlson, 1982
  - Sprague and Watson, 1993
  - Sprague and Watson, 1996
  - Thierauf, 1982
  - Thierauf, 1988
  - Turban, 1995
  - Young, 1989

- Executive Information Systems
  - Bird, 1991
  - Burkan, 1991
  - Paller, 1990
  - Rockart and DeLong, 1988
  - Thierauf, 1991
  - Watson and Houdeshel, 1992

- Expert Systems
  - Gonzalez and Dankel, 1993
  - Klein and Methlie, 1995
  - Mockler, 1989a
  - Mockler, 1989b
- Group Decision Support Systems (GDSS)
  - Bostrom et. al., 1992
  - Bui, 1987
  - Coleman and Khanna, 1995 (eds.)
  - Jessup and Valacich, 1993
  - Thierauf, 1989
- Teamware
  - Baecker, 1993
  - Johansen, 1988
  - Johansen et al., 1991
  - Opper and Fersko-Weiss, 1992

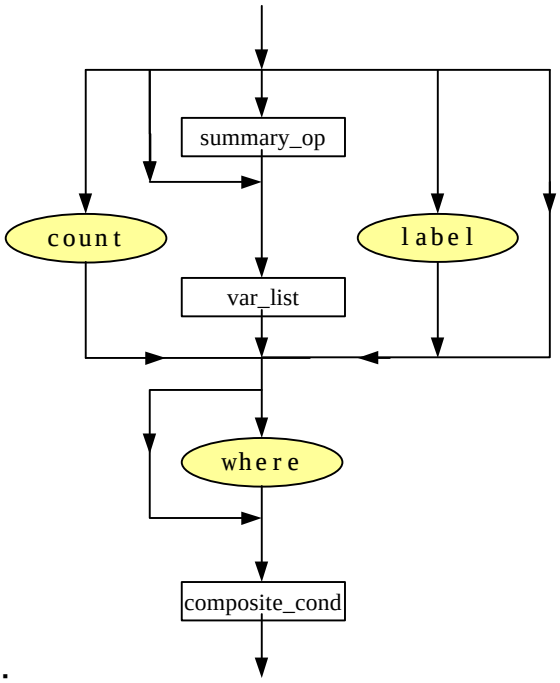


3-way TANGRAM - Basic terms

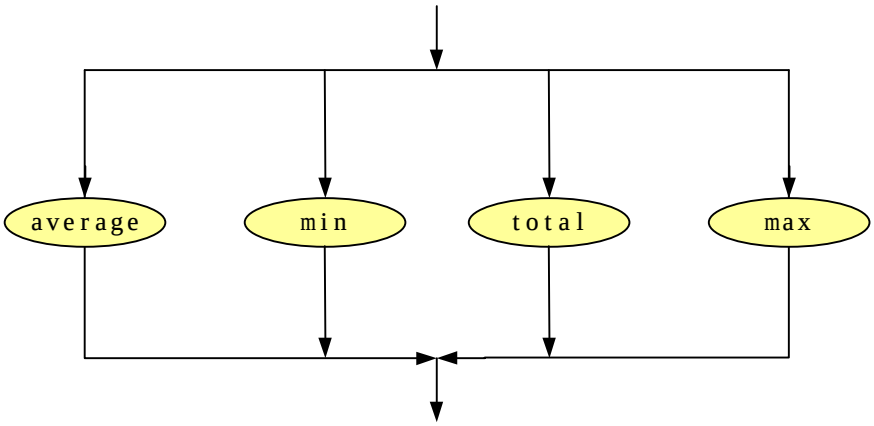

CODEWORK
DBMS
Flat file
Office, Windows Apps
Clipboard
XML
Documented TANGRAM format
Template driven HTML
Excel
Template driven ASCII
LaTeX
PostScript
Acrobat
TANGRAM hypercube formats
Offered by Tangram
Readily available

MG,VT 01/2001

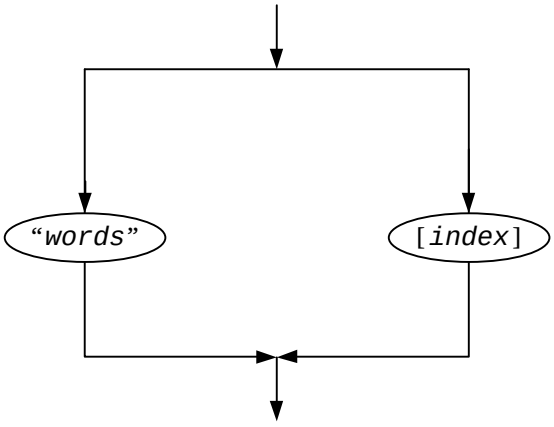
query:



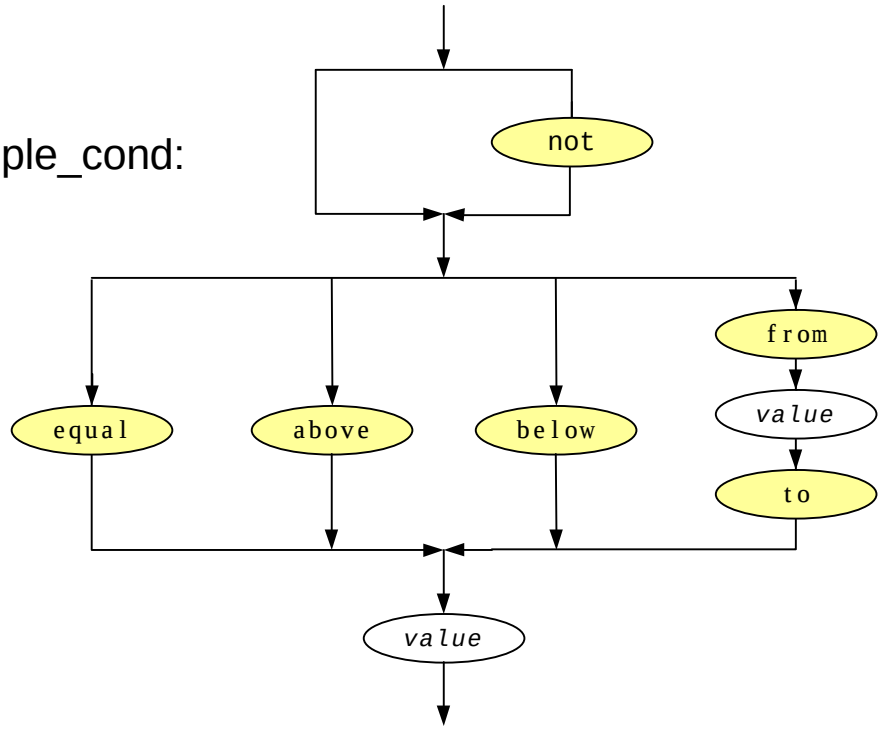
summary_op:



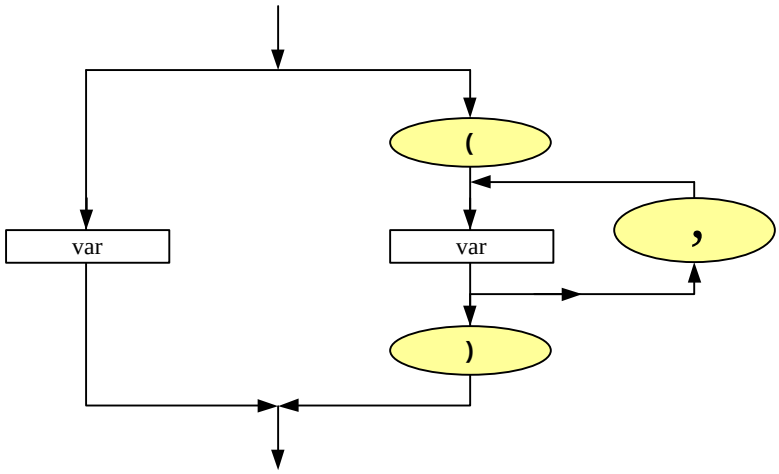
var:



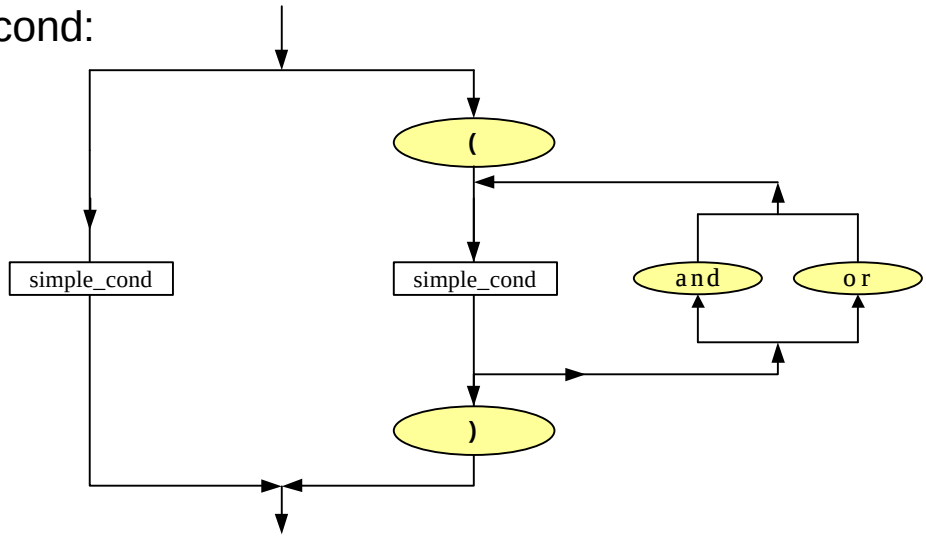
simple_cond:



var_list:



composite_cond:



CODEWORK

Analysis of label structure in sample cube MSALES (monthly tractor sales).