AIMMS Language Reference - Read and Write Statement

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com.

AIMMS B.V.
Diakenhuisweg 29-35
2033 AP Haarlem
The Netherlands
Tel.: +31 23 5511512

AIMMS Inc.
11711 SE 8th Street
Suite 303
Bellevue, WA 98005
USA
Tel.: +1 425 458 4024

AIMMS Pte. Ltd.
55 Market Street #10-00
Singapore 048941
Tel.: +65 6521 2827

AIMMS
SOHO Fuxing Plaza No.388
Building D-71, Level 3
Madang Road, Huangpu District
Shanghai 200025
China
Tel.: ++86 21 5309 8733

Email: info@aimms.com
WWW: www.aimms.com

Chapter 26

The READ and WRITE Statements

Dynamic data transfer

In order to help you separate the model description and its input and output data, AIMMS offers the READ and WRITE statements for dynamic data transfer between your modeling application and external data sources such as

- *text data files*, and
- *database tables* in external ODBC-compliant databases.

This chapter

This chapter first introduces the READ and WRITE statements in the form of an extended example. Subsequently, their semantics are presented in full detail including issues such as filtering, domain checking, and slicing.

26.1 A basic example

Getting started

The aim of this section is to give you an overview of the READ and WRITE statements through a short illustrative example. It shows how to read data from and write data to text files and database tables. It is based on the familiar transport problem with the following input data:

- the set Cities,
- the compound set Routes from Cities to Cities,
- the parameters Supply(i) and Demand(i) for each city i, and
- the parameters Distance(i,j) and TransportCost(i,j) for each route between two cities i and j.

For the sake of simplicity, it is assumed that there is only a single output, the actual Transport(i,j) along each route.

Format of input data

The input data can be conveniently given in the form of tables. One for the identifiers defined over a single city like Supply and Demand, and the other for the identifiers defined over a tuple of cities like Distance and TransportCost. These tables can be provided in the form of text files as in Table 26.1 (format explained in Section 28.3). Alternatively, the data can be obtained from particular tables in a database. This example assumes the following database tables exist:

- CityData for the one-dimensional parameters, and

- RouteData for the two-dimensional parameters.

```
COMPOSITE TABLE
    Cities      Supply  Demand
!   ----------  ------  ------
    Amsterdam     50
    Rotterdam    100
    Antwerp       75      25
    Berlin                125
    Paris                  75
;
```

```
COMPOSITE TABLE
    i          j          Distance  TransportCost
!   ---------  ---------  --------  -------------
    Amsterdam  Rotterdam       85        1.00
    Amsterdam  Antwerp        170        2.50
    Amsterdam  Berlin         660       10.00
    Amsterdam  Paris          510        8.25
    Rotterdam  Antwerp        100        1.20
    Rotterdam  Berlin         700       10.00
    Rotterdam  Paris          440        7.50
    Antwerp    Berlin         725       11.00
    Antwerp    Paris          340        5.00
    Berlin     Paris         1050       17.50
;
```

Table 26.1: Example data set for the transport model

26.1.1 Simple data transfer

Simple data initialization

The simplest use of the READ statement is to initialize data from a fixed name text data file, or a database table. To read all the data from each source, the following groups of statements will suffice

```
read from file  "transport.inp" ;

read from table CityData;
read from table RouteData;
```

Such statements are typically found in the body of the predefined procedure MainInitialization.

Reading identifier selections

When a data source also contains data for identifiers that are of no interest to your particular application (but may be to others), AIMMS allows you to restrict the data transfer to a specific selection of identifiers in that data source. For instance, the following READ statement will only read the identifiers Distance and TransportCost, not changing the current contents of the AIMMS identifiers Supply and Demand.

```
read Distance, TransportCost from file "transport.inp" ;
```

Similar identifier selections are possible when reading from a database table.

Writing the solution

After your model has computed the optimal transport, you may want to write the solution Transport(i,j) to an text output file for future reference. You can do this by calling the WRITE statement, which has equivalent syntax to the READ statement. The transfer of Transport(i,j) to the file transport.out is accomplished by the following WRITE statement.

```
write Transport to file "transport.out" ;
```

If you omit an identifier selection, AIMMS will write all model data to the file. When writing to a database table, AIMMS can of course only transfer data for those identifiers that are known in the table that you are writing to.

File name need not be explicit

File data transfer is not restricted to files with a fixed name. To choose the name of the data file either during execution or from within the end-user interface, you have several options:

- replace the filename string in the READ and WRITE statements with a string-valued parameter holding the filename, or
- use a File identifier (for text files only).

26.1.2 Set initialization and domain checking

Domain restrictions

When you are reading the initial data of the transport model from an external data source several situations can occur:

- you just want to initialize the set Cities from the data source,
- the set Cities has already been initialized, and you want to retrieve the parametric data for existing cities only, or
- the set Cities has already been initialized, but you want to extend it on the basis of the data read from the external data source.

The READ statements

The following statements impose domain restrictions on the READ statement.

```
read Cities
     from file "transport.inp" ;

read Supply, Demand
     from file "transport.inp"
     filtering i ;

read Supply, Demand
     from file "transport.inp" ;
```

Initializing sets

The first READ statement is a straightforward initialization of the set Cities. By default, AIMMS reads in replace mode, which implies that any previous contents of the set Cities is overwritten.

Domain checking

The second READ statement assumes that the set Cities has already been initialized. From all entries of the identifiers Supply and Demand it will only read those which correspond to existing elements in the set Cities, and skip over the data from the remaining entries.

Extending domain sets

The third READ statement differs from the second in that the clause 'FILTERING i' has been omitted. As a result, AIMMS will not reject data that does not correspond to an existing label in the set Cities, but will read all available Supply and Demand data, and extend the set Cities accordingly.

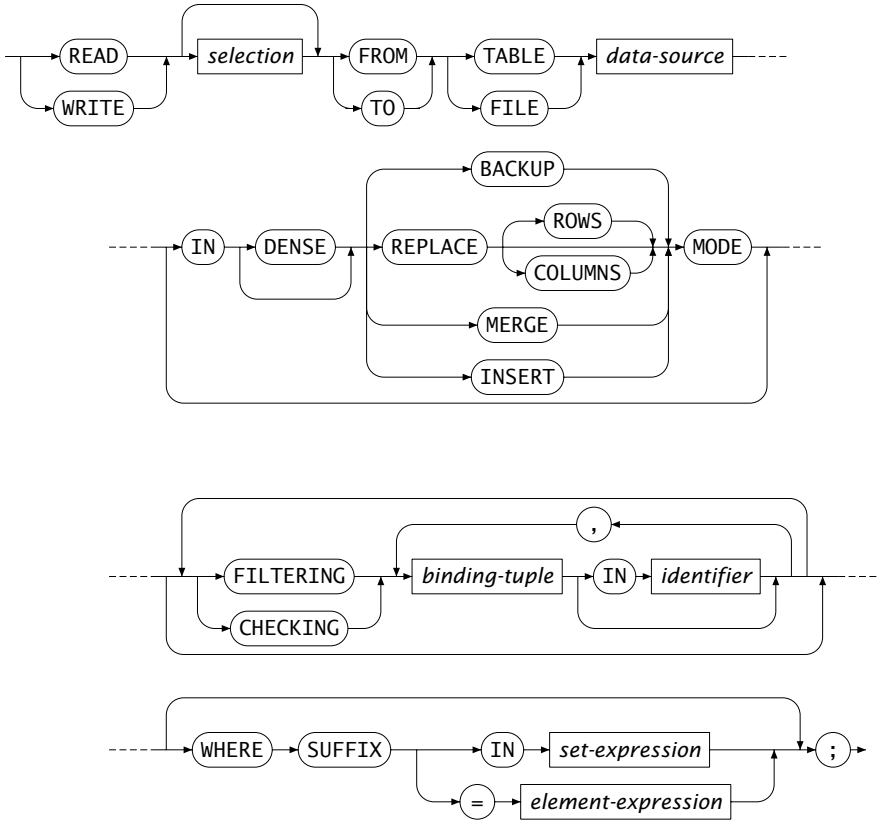
26.2 Syntax of the READ and WRITE statements

READ and WRITE statements

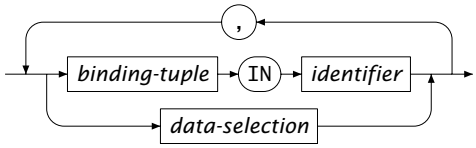
In READ and WRITE statement you can specify the data source type, what data will be transferred, and in what mode. The syntax of the statements reflect these aspects.

Syntax

read-write-statement :



selection :



Data sources

The data source of a READ or WRITE statement in AIMMS can be either

- a File represented by either
  - a File identifier,
  - a string constant, or
  - a scalar string reference,
- a TABLE represented by either
  - a DatabaseTable identifier,
  - an element parameter with a range that is a subset of the predeclared set AllDatabaseTables

Strings for file data sources refer either to an absolute path or to a relative path. All relative paths are taken relative to the project directory.

Examples

Assuming that UserSelectedFile is a File identifier, and UserFilename a string parameter, then the following statements illustrate the use of strings and File identifiers.

```
read from file "C:\Data\Transport\initial.dat" ;
read from file "data\initial.dat" ;
read from file UserFileName ;
read from file UserSelectedFile ;
```

Specifying a selection

The *selection* in a READ or WRITE statement determines which data you want to transfer from or to a text file, or database table. A selection is a list of references to sets, parameters, variables and constraints. During a WRITE statement, AIMMS accepts certain restrictions on each reference to restrict the amount of data written (as explained below). Note, however, that AIMMS does not accept all types of restrictions which are syntactically allowed by the syntax diagram of the READ and WRITE statements.

Default selection

If you do not specify a selection during a READ statement, AIMMS will transfer the data of all identifiers stored in the table or file that can be mapped onto identifiers in your model. If you do not specify a selection for a WRITE statement to a text file, all identifiers declared in your model will be written. When writing to a database table, AIMMS will write data for all columns in the table as long as they can be mapped onto AIMMS identifiers.

Filtering the selection

You can apply the following filtering qualifiers on READ and WRITE statements to restrict the data selection:

- the FILTERING or CHECKING clauses restrict the domain of all transferred data in both the READ and WRITE statements, and
- an arbitrary logical condition can be imposed on each individual parameter and variable in a WRITE statement.

FILTERING versus CHECKING

You can use both the FILTERING and CHECKING clause to restrict the tuples for which data is transferred between a data source and AIMMS. During a WRITE statement there is no difference in semantics, and you can use both clauses interchangeably. During a READ statement, however, the FILTERING clause will skip over all data outside of the filtering domain, whereas the CHECKING clause will issue a runtime error when the data source contains data outside of the filtering domain. This is useful feature for catching typing errors in text data files.

Examples

The following examples illustrate filtering and the use of logical conditions imposed on index domains.

```
read Distance(i,j) from table RouteTable
     filtering i in SourceCities, (i,j) in Routes;
```

```
write Transport( (i,j) | Sum(k, Transport(i,k)) > MinimumTransport )
      to table RouteTable ;
```

Advanced filtering on records

If you need more advanced filtering on the records in a database table, you can use the database to perform this for you. You can

- define *views* to create temporary tables when the filtering is based on a non-parameterized condition, or
- use *stored procedures* with arguments to create temporary tables when the filtering is based on a parameterized condition.

The resulting tables can then be read using a simple form of the READ statement.

Merge, replace or backup mode

AIMMS allows you to transfer data from and to a file or a database table in *merge* mode, *replace* mode or *insert* mode. If you have not selected a mode in either a READ or WRITE statement, AIMMS will transfer the data in replace mode by default. When you are writing data to a text data file, AIMMS also supports a *backup* mode. The *insert* mode can speed up writing to databases.

Reading in merge mode

When AIMMS reads data in merge mode, it will overwrite existing elements for all read identifiers, and add new elements as necessary. It is important to remember that in this mode, if there is no data read for some of the existing elements, they keep their current value.

Writing in merge mode

When AIMMS writes data in merge mode, the semantics is dependent on the type of the data source.

- If the data source is a text file, AIMMS will *append* the newly written data to the end of the file.
- If the data source is a database table, AIMMS will merge the new values into the existing values, creating new records as necessary.

Reading in replace mode

When AIMMS reads data in replace mode, it will empty the existing data of all identifiers in the identifier selection, and then read in the new data.

Writing in replace mode

When AIMMS writes data in replace mode, the semantics is again dependent on the type of the data source.

- If the data source is a text file, AIMMS will *overwrite the entire contents* of the file with the newly written data. Thus, if the file also contained data for identifiers that are not part of the current identifier selection, their data is lost by the WRITE statement.
- If the data source is a database table, AIMMS will either empty all columns in the table that are mapped onto identifiers in the identifier selection

(default, REPLACE COLUMNS mode), or will remove all records in the table not written by this write statement (REPLACE ROWS mode). The REPLACE COLUMNS and REPLACE ROWS modes are discussed in more detail in Section 27.3).

Writing in insert mode

Writing in insert mode is only applicable when writing to databases. Essentially, what it does is writing the selected data to a database table using SQL *INSERT* statements. In other words, it expects that the selection of the data that you write to the table doesn't match any existing primary keys in the database table. If it does, AIMMS will raise an error message about duplicate keys being written. Functionally, the insert mode is equivalent to the replace rows mode, with the non-existing primary keys restriction. Especially when writing to database tables which already contain a lot of rows, the speed advantage of the insert mode becomes more visible.

Writing in backup mode

When you are transferring data to a text file, AIMMS supports writing in backup mode in addition to the merge and replace modes. The backup mode lets you write out files which can serve as a text backup to a (binary) AIMMS case file. When writing in backup mode, AIMMS

- skips all identifiers on the identifier list which possess a nonempty definition (and, consequently, cannot be read in from a datafile),
- skips all identifiers for which the property NoSave has been set, and
- writes the contents of all remaining identifiers in such an order that, upon reading the data from the file, all domain sets are read before any identifiers defined over such domain sets.

Backup mode is not supported during a READ statement, or when writing to a database.

Writing data in a dense mode

Data in AIMMS is stored for non-default values only, and, by default, AIMMS only writes these non-default values to a database. In order to write the default values as well to the database table at hand, you can add the *dense* keyword before most of the WRITE modes discussed above. This will cause AIMMS to write all possible values, including the defaults, for all tuple combinations considered in the WRITE statement. Care should be taken that writing in *dense* mode does not lead to an excessive amount of records being stored in the database. The mode combination *merge* and *dense* is not allowed, because it is ambiguous whether or not a non-default entry in the database should be overwritten by a default value of AIMMS.

Replacing sets

Whenever elements in a domain set have been removed by a READ statement in replace mode, AIMMS will *not* cleanup all identifiers defined over that domain. Instead, it will leave it up to you to use the CLEANUP statement to remove the inactive data that may have been created.

Domain filtering

For every READ and WRITE statement you can indicate whether or not you want domain filtering to take place during the data transfer. If you want domain filtering to be active, you must indicate the list of indices, or domain conditions to be filtered in either a FILTERING of CHECKING clause. In case of ambiguity which index position in a parameter you want to have filtered you must specify indices in the set or parameter reference.

Example

The following READ statements are not accepted because both Routes and Distance are defined over Cities × Cities, and it is unclear to which position the filtered index i refers.

```
read Routes   from table RouteTable filtering i ;
read Distance from table RouteTable filtering i ;
```

This ambiguity can be resolved by explicitly adding the relevant indices as follows.

```
read (i,j) in Routes from table RouteTable filtering i ;
read Distance(i,j)   from table RouteTable filtering i ;
```

Semantics of domain filtering

When you have activated domain filtering on an index or index tuple, AIMMS will limit the transfer of data dependent on further index restrictions.

- During a READ statement only the data elements for which the value of the given index (tuple) lies within the specified set are transfered. If no further index restriction has been specified, transfer will take place for all elements of the corresponding domain set.
- During a WRITE statement only those data elements are transferred for which the index (tuple) is contained in the AIMMS set given in the (optional) IN clause. If no set has been specified, and the data source is a database table, the transfer is restricted to only those tuples that are already present in the table. When the data source is a text file the latter type of domain filtering is not meaningful and therefore ignored by AIMMS.

READ example

In the following two READ statements the data transfer for elements associated with i and (i,j), respectively, is further restricted through the use of the sets SourceCities and Routes.

```
read Distance(i,j) from table RouteTable filtering i in SourceCities ;
read Distance(i,j) from table RouteTable filtering (i,j) in Routes ;
```

WRITE example

In the following two WRITE statements, the values of the variable Transport(i,j) are written to the database table RouteTable for those tuples that lie in the AIMMS set SelectedRoutes, or for which records in the table RouteTable are already present, respectively.

```
write Transport(i,j) to table RouteTable filtering (i,j) in SelectedRoutes ;
write Transport(i,j) to table RouteTable filtering (i,j) ;
```

The FILTERING clause in the latter WRITE statement would have been ignored by AIMMS when the data source was a text data file.

Writing selected suffices using the WHERE clause

Using the WHERE clause of the WRITE statement you can instruct AIMMS, for all identifiers in the identifier selection, to write the data of either a specified suffix or a set of suffices to file, rather than their level values. The WHERE clause can only be specified during a WRITE statement to a FILE, and the corresponding set or element expression must refer to a subset of, or element in, the predefined set AllSuffixNames.

Example

The following WRITE statement will write the values of the .Violation suffix of to the file ViolationsReport.txt for all variables in the project.

```
write AllVariables to file "ViolationsReport.txt" where suffix = 'Violation';
```