Overview of commands

Part 1

s-commands

The commands *type, delete type, create, update, let, derive, delete* and *kill* work within the SecondoREPLAY Client different to the SecondoTTYCS client, if a cluster has been configured with at least one Node and started successfully.

If one of these commands is entered at the started Cluster, these commands are executed on the master and on all nodes.

The following command sets the object last_name on the Master and on all active nodes:

let last_name = "Kunsmann"

Accordingly, this object is deleted with the following command on the master and all the nodes:

delete last_name

The newly added commands with a *s* as prefix work with the started cluster like commands without this prefix within the SecondoTTYCS Client. An execution is carried out only on the Master.

Following these new commands are briefly presented, which are explained on the basis of *SecondoManual*. There you can find more details about the commands.

stype *<identifier> = <type expression>*

Creating a new type named <identifier> and the expression <type expression>.

sdelete type *<identifier>*

Deletes a user defined type named <identifier>.

screate *<identifier> : <type expression>*

Creating an object with the name <identifier> and the type <type expression>. The value is still undefined.

supdate *<identifier> := <value expression>*

The existing object <identifier> is assigned the result of the expression <value expression>.

slet *<identifier> = <value expression>*

It will create a new object named <identifier>. This is assigned the result of the expression <value expression>. If an object already exists, there is an error and the command is stopped without changing. The type of the object is determined automatically on the base of the result of the expression and set accordingly.

sderive <*identifier*> = <*value expression*>

This is a variant of the let-commands. Hereby an object is created, which has a different object as input. These objects do not have an external list representation, as it is the case with indexes. When restoring a database, this object is automatically reconstructed.

sdelete <*identifier*>

Deletes an object with the name <identifier>.

skill <*identifier*>

Removes an object named <identifier> from the catalog of the database, but the data structures remain here. This can be useful when an object can not be deleted via the normal delete command, because of a corrupt data structure. In normal daily operation, however, delete should be used.

Overview Command (Replay-Version with Execution Destination)

Basic Commands (MASTER + NODES)	Inquiries (MASTER)
type <identifier> = <type expression> delete type <identifier> create <identifier> : <type expression> update <identifier> := <value expression> let <identifier> = <value expression> derive <identifier> = <value expression> delete <identifier> kill <identifier> if <p> then <command> [else <command>] endif while <p> do <command> endwhile { <command> [| <command>]*] } {{ <command> [| <command>]*] }}	list type constructors list operators list algebras list algebra <identifier> list databases list types list objects
Basic Commands (MASTER)	**Transactions (MASTER + NODES)**
query <value expression> stype <identifier> = <type expression> sdelete type <identifier> screate <identifier> : <type expression> supdate <identifier> := <value expression> slet <identifier> = <value expression> sderive <identifier> = <value expression> sdelete <identifier> skill <identifier>	begin transaction commit transaction abort transaction
Databases (MASTER + NODES)	
create database <identifier> delete database <identifier> open database <identifier> close database <identifier>	
Import and Export (MASTER)	
save database to <file> restore database <identifier> from <file> restore <identifier> from <file>	

Part 2

Note:
All parameters "filename" can be an relative path, starts from the location of the bin-directory of Secondo, or an absolute path.

replayOSMImport (filename[,replay Import Mode])

This command splits an OSM file into disjoint fileparts which are distributed on the active nodes and are imported there. The Name of the OSM file is handed over the parameter *filename*.

Over the optional parameter *replayImportMode* is specified whether you want to import on all nodes of the entire dataset (*Replication*), or whether the data should be distributed disjunctive on the nodes (*Partitioning*). If the parameter is omitted, the default value from the configuration file SecondoConfig.ini is used. The corresponding parameter is called *ReplayImportMode*.

Dependencies:
On Master and all Nodes the **OsmAlgebra** must be installed.

Example:
Secondo => replayOSMImport(MyData/osm/Monaco-latest.osm)

replayCSVImport(filename, relName, headersize, comment, separator, uses_quotes, multiline[,replayImportMode])

The distributed import of a CSV file results over this command. With the parameter *filename* the corresponding file name is set. Is *relName* when importing the name of the relation defined that as a suffix in addition to the underscore nor the Number of the split file is attached. The definition of the tuple is carried out via the *header* the CSV file. The number of lines of the header within the file is defined by *headersize*. With the parameter *comment* is set the comment sign. All lines which are starting with this sign are ignored during import. With the *separator* the delimiter of the single data record is fixed. Here is an empty string passed as a parameter the comma (,) is used as a delimiter. Can the delimiter itself occur within the records, the data must comply with Quotes are enclosed and the parameters are *uses_quotes* set to *TRUE*. Can be contained multiline records they are enclosed in quotation marks, and the parameters *mulitline* be set to *TRUE*.

Over the optional parameter *replayImportMode* is specified whether you want to import on all nodes of the entire dataset (*Replication*), or whether the data should be distributed disjunctive on the nodes (*Partitioning*). If the parameter is omitted, the default value from the configuration file SecondoConfig.ini is used. The corresponding parameter is called *ReplayImportMode*.

Dependencies:
On all Nodes the **ImExAlgebra** must be installed.

Example:
Secondo => replayCSVImport(MyData/csv/Invoices.csv,mtcsv,1,,,FALSE,FALSE)

replaySHPImport(filename, objName, geoName[,replayImportMode])

Files in Shape (SHP) format can be imported using this command on the active node. For this purpose, the shapefile and the associated dBASE (DBF) file is split into disjoint files on the master. These are transferred to the active nodes and imported there. The generated Relation names are created based on the parameter *objName*. This is a suffix for each imported file will be added (_1, _2, etc.). The parameter *geoName* the attribute name is set, which will contain the geometry.

Over the optional parameter *replayImportMode* is specified whether you want to import on all nodes of the entire dataset (*Replication*), or whether the data should be distributed disjunctive on the nodes (*Partitioning*). If the parameter is omitted, the default value from the configuration file SecondoConfig.ini is used. The corresponding parameter is called *ReplayImportMode*.

Dependencies:
On all Nodes the **ImExAlgebra** must be installed.

Example:
Secondo => replaySHPImport(MyData/shp/railways.shp,Railways,GeoData)

replayDBLPImport(filename[,replayImportMode])

This one has the possibility the current DBLP file distributed to import on the active nodes. The master splits the file into this disjoint files. From each of these sub-files are now relations files (*Document_No, Author_No, Authordoc_No and Keyword_No*) created in CSV format. The file name is appended a suffix that reflects the number of the split DBLP file. The relations names which are created in the database by importing, to filenames with the corresponding suffix.

Over the optional parameter *replayImportMode* is specified whether you want to import on all nodes of the entire dataset (*Replication*), or whether the data should be distributed disjunctive on the nodes (*Partitioning*). If the parameter is omitted, the default value from the configuration file SecondoConfig.ini is used. The corresponding parameter is called *ReplayImportMode*.

Dependencies:
On all Nodes the **ImExAlgebra** must be installed.
On Master the **Dblp2Secondo Converter** must be installed (it will implicitly compiled if not installed).

Example:
Secondo => replayDBLPImport(MyData/dblp/dblp.xml)

replayIMGImport(startDirectory, recursiveLevel, relationName[,replayImportMode])

Specifies the nodes a relation named *relationName* to which the images from the Directory *startDirectory* contains. About the *recursiveLevel* parameter determines how many levels (subfolders) should be considered. A value of *1* would consider only the directory *startDirectory*. The value *2* additionally all subfolders. The value *3* for all subfolders in each case their subfolders etc.

The relationship also includes the images in binary format. These can use the JavaGui Secondo be represented graphically, provided that there is a viewer for the appropriate format.

Currently the following formats are supported:

• JPG (Joint Photographics Expert Group)

• GIF (Graphics Interchange Format)

• TIF (Tagged Image File Format)

• BMP (Windows Bitmap)

• PNG (Portable Network Graphics)

Note that no check is made as to whether it is indeed graphics corresponding format in the files. The transfer list is determined purely on the basis of the file extension.

The optional parameter *replayImportMode* specifies whether all nodes on a relation with all the pictures of the transfer list should be created (*Replication*), or if the images are to be distributed to the disjunctive node (*Partitioning*). If the parameter is omitted, the default value from the configuration file *SecondoConfig.ini* is used. The corresponding parameter is called *ReplayImportMode*.

Dependencies:
-

Example:
Secondo => replayIMGImport(MyData/images,3,mtimages)

shareFile(filename, cpDestPath)

Transferred the file *filename* (absolute or relative path possible) from the master to all active nodes. On the Nodes, the file is in your home directory the user within the folder *secondo-databases/filetransfers/[PID]* filed. [PID] is the process ID with which the master is connected to the corresponding node. Is within the SecondoConfig.ini a different base directory for storing the database defined, it will be used *($SECONDO_HOME/filetransfers/[PID]*).

The second parameter is used to the transferred file in the directory *cpDestPath* copy. For this, the user must have write access to this directory have on all nodes. An automatic system of directories does not take place.

Dependencies:
-

Example:
Secondo => shareFile(MyData/share/Secondo-mod.pdf,/home/cluster/secondo/Data/transfer)