Author a command line tool (CLI) in a JVM-based language such as Java or Scala that can watch for, extract, transform, and load tabular data discovered in a filesystem folder into a relational database (e.g. SQLite).

Imagine a laboratory instrument is used at various times throughout the day and when the instrument is used, it writes a regular CSV file to a single filesystem folder with a unique filename prefixed with the ISO8601 date timestamp.

Your program must continuously watch the filesystem folder for the appearance of these files and load them into a relational database.

Please also provide a bootstrap script for initializing an empty database with the necessary schema/tables.

Include a written description of no more than 500 words on how robust your solution is and what features you need to implement if you are asked to prove an audit trail of all data.

Also describe how your application may need to be modified or deployed if the laboratory buys 10 instruments and all instruments are configured to write to different filesystem folders, but you are constrained to using a single relational database.

This InstrumentDataParser package accepts as arguments either directories (using the "-d <path\to\directory>" option) or text files containing lists of directories ("-f <file-path.txt>"), as well as combinations of multiples of the two (e.g. "-d 'C:\Users\Me\Documents\Data Files' -f 'file1.txt' 'file2.txt'"). It combines the directory paths from each of these sources and registers event watchers to trigger when new CSV files are added. When that happens, the text of those new files is parsed to generate a list of field names, and each row of data is inserted into a SQLite database using prepared statements. If the database does not exist, it is created; likewise, the Results table is created if necessary. A trigger is created to add the date and time to each record as it is inserted, and the SourceFile field stores the file path from which each record was extracted; if auditing becomes necessary, this should be very useful, but it may be wise to add a "LastUpdated" field along with an "AFTER UPDATE" trigger to track the date and time of any modifications to each record.

No standalone bootstrap scripts have been provided since the application itself creates the database file automatically in the project folder. If a different database file location is desired, the *dbfile* string in the DataMonitor class can be modified accordingly.

To use any RDBMS besides SQLite, a corresponding DataAdapter class should be added.

More instruments can easily be accommodated by adding their output directories to the command line arguments as specified above, provided their data fields exist as columns in the Results table. The Results table is created with fields matching the headings in the example CSV files (along with a few others, mentioned above). To allow for insertion of data from instruments with different output fields, modify the Results table to contain the union of all field sets; the prepared statement will insert the data into the correct columns.

Note that this program only adds records from files as they are added to the folder. Any existing files will be ignored unless they are removed from the directory and added again (e.g., cut, paste into different folder, then drag them back to the original location). Also note that this implementation assumes files are complete when created and will not be modified afterward, for example, if files are automatically copied to the target directory when a result set is completed. It may behave unpredictably if results are written directly to the file individually since the file contents may be incomplete when the file creation event is triggered.