KNIME Analytics Platform: Tips & Tricks for Workflow Developers

Martyna Pawletta

Data Scientist in Life Sciences

Tobias Kötter

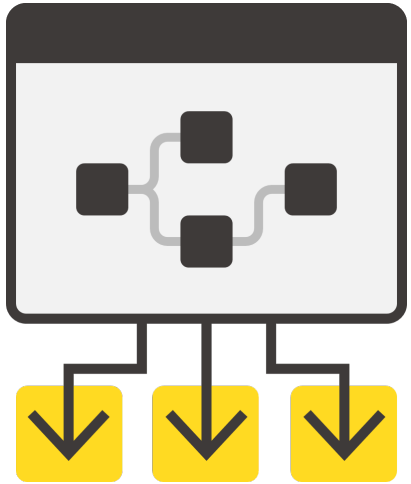
Product Management

April 19, 2023

Do you recognize yourself in this situation?

You have now created your workflow, and are ready to put it into use! But…

1. You feel like your workflow is slower than you expected
2. Your workflow is big and confusing, and maintenance has become an issue
3. You run into frequent memory errors
4. You think it needs to be a bit more secure



KNIME Workflow Building – Best Practices

- Creating a KNIME workflow is like telling a news story.

- To tell a news story well, you must have:
  - A defined aim/goal
  - A clear introduction
  - Concise – avoiding being repetitive and long winded
  - An adequate usage of space – not too long, not too little

KNIME Workflow Building – Best Practices

- Creating a KNIME workflow is like telling a news story.

- To ~~tell~~ develop a good a ~~news story well~~ KNIME workflow, you must have:
  - A defined aim/goal (A well-defined workflow scope)
  - A clear introduction (Efficient reading of input data)
  - Concise – avoiding being repetitive and long winded (Appropriate selection of nodes)
  - An adequate usage of space – not too long, not too little (Careful consideration about computational resources)

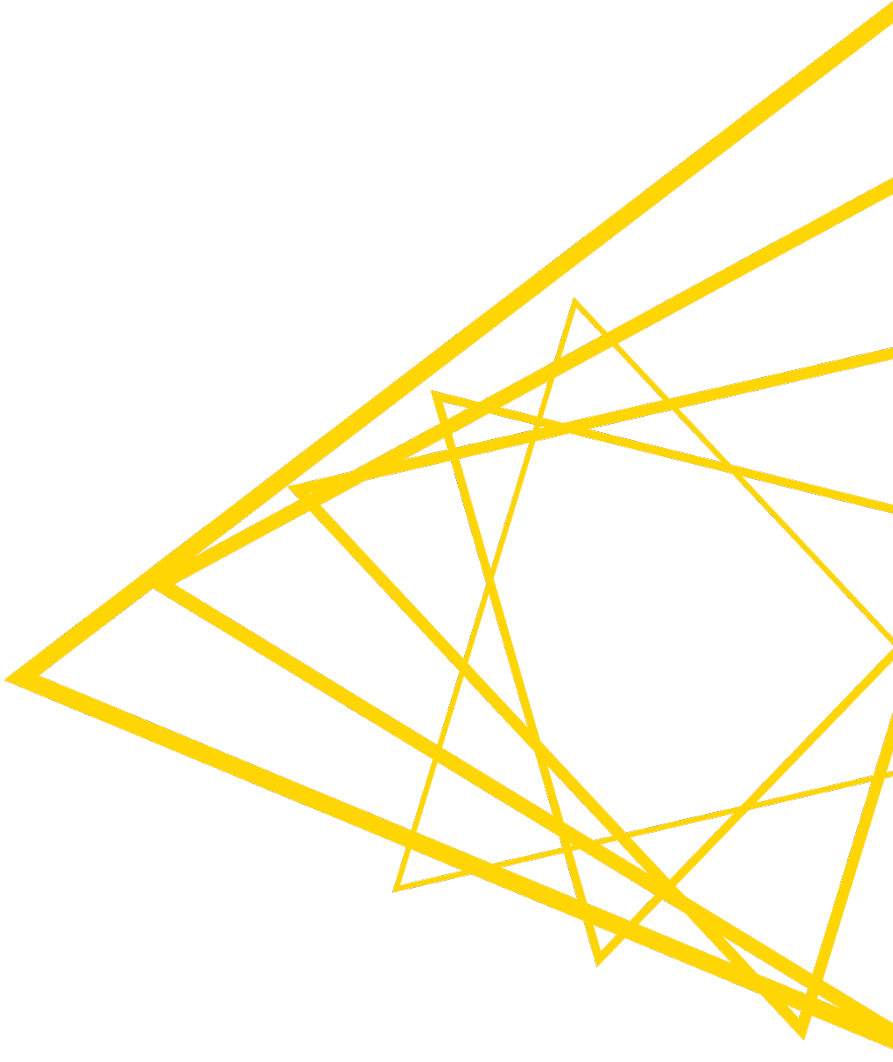
The Pillars of a Good Workflow

Reusable

Secure

Efficient

Reusability

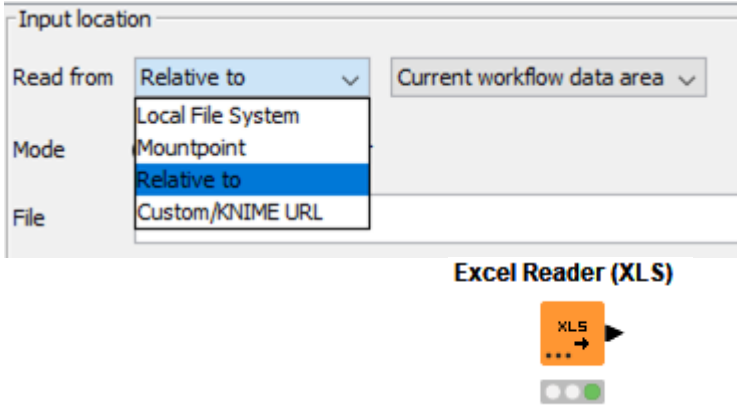


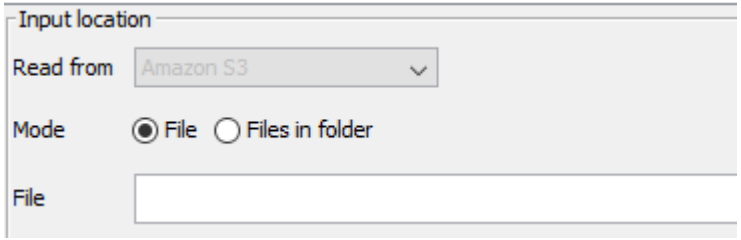
Build in Flexible Data Access

- Make sure your workflow can work everywhere, not only on your local machine:
  - Make use of relative paths instead of your local path so that workflow will work everywhere.
  - When connecting to remote drives, make use of the appropriate connector nodes to make your workflow flexible
- Make use of configuration and widget nodes to make your workflow flexible.
  - For instance: do not hardcode your credentials in DB connector nodes – make use of the credentials configuration node instead.

Accessing Files from Anywhere

- Convenience File Systems
  - Local File System
  - Mountpoint
  - Relative to (workflow, mountpoint, workflow data area)
    - *Key to moving data with workflows/groups*
  - Custom/KNIME URL
- Connected File Systems
  - Microsoft Azure
  - Google
  - Amazon
  - Databricks
  - BigData file systems (HDFS, HTTPFS, …)
  - On-premise (e.g., SSH, FTP, …)
- File handling nodes with dynamic input port

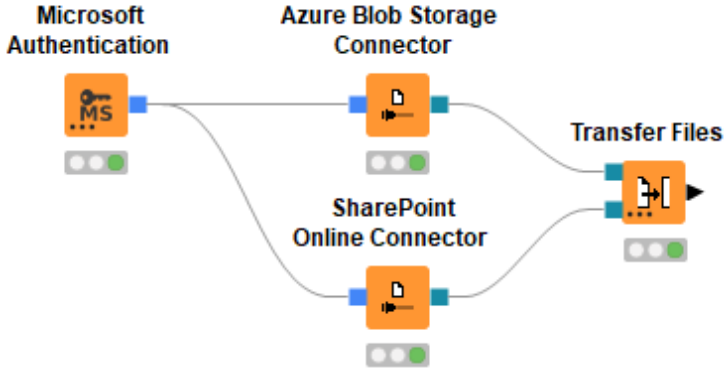




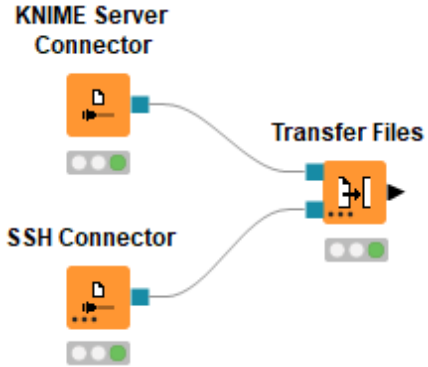
File Handling Framework - Flexibility

Local file system and KNIME mountpoints

Cross cloud environments



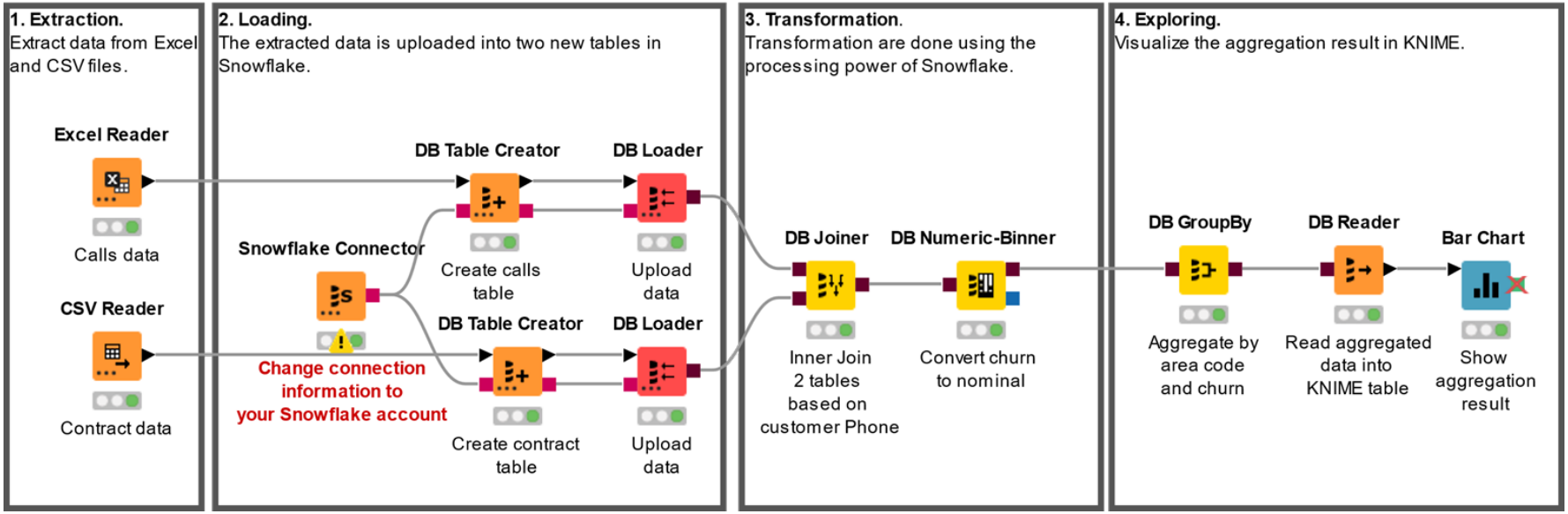
Same cloud environment



On-premise

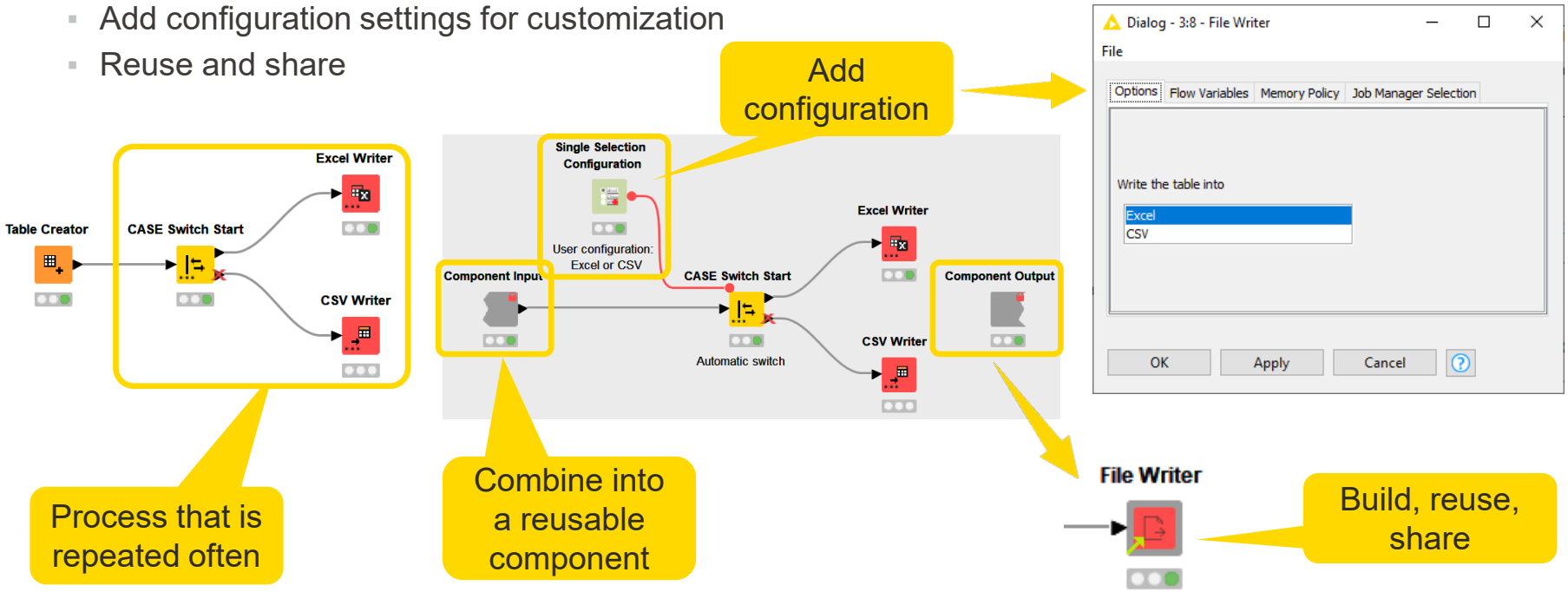
Reusability via Documentation

Preparing and Exploring Customer Data with KNIME and Snowflake (ELT)
This workflow is an example of how to read different data files in KNIME and upload it to Snowflake. Once uploaded the data is transformed and analysed within Snowflake and the result is visualized in KNIME.
Example data provided from kaggle: https://www.kaggle.com/becksddf/churn-in-telecoms-dataset



Reusability via Components

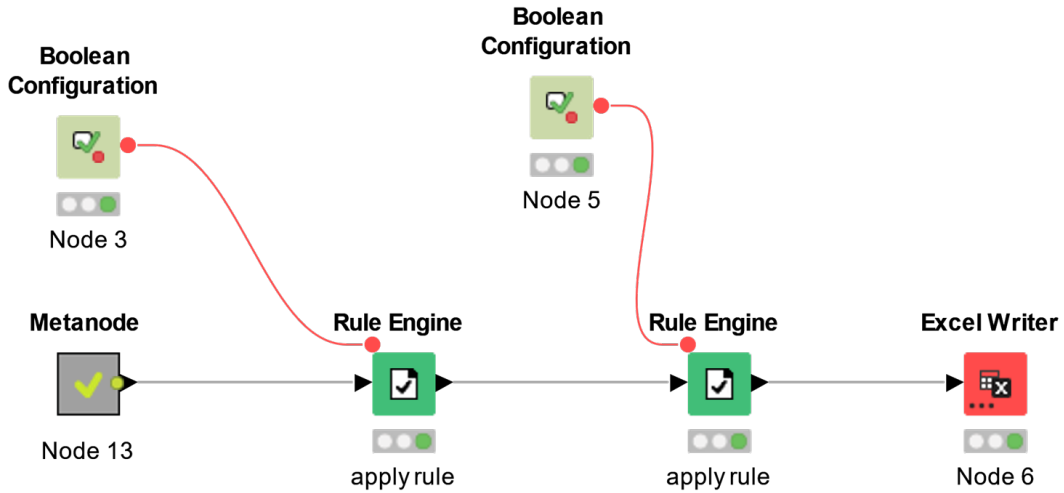
- Logic blocks and repeatable processes can be written once and reused
- In KNIME Analytics Platform – by means of reusable components
  - Wrap a process into a component
  - Add configuration settings for customization
  - Reuse and share



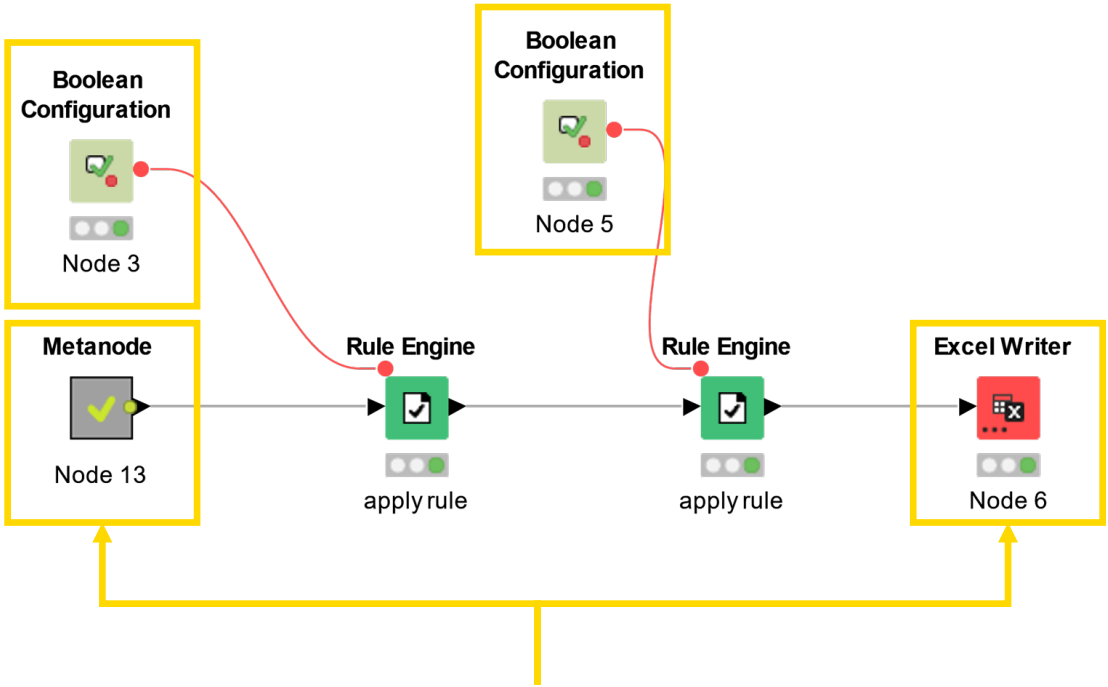
Benefits of Using Components within an Organization

- Standardization
  - Your own organization's best practices and procedures can be implemented as components
  - Component-driven design & development helps promote creation of reusable processes with a single responsibility. For example…
    - Organization-specific database connectors
    - Standardized ETL cleanup operations
    - Standardized logging operations
    - Model API interfaces (i.e., schema validation and transformation of JSON input/output)
- Adherence to modern testing and design principles
  - Components help teams follow the Don't Repeat Yourself (DRY) principle
  - Components are small, independently testable units of functionality.
- Workflow organization
  - Components can be nested inside other components, allowing complex workflows to be comprised of small, maintainable parts.

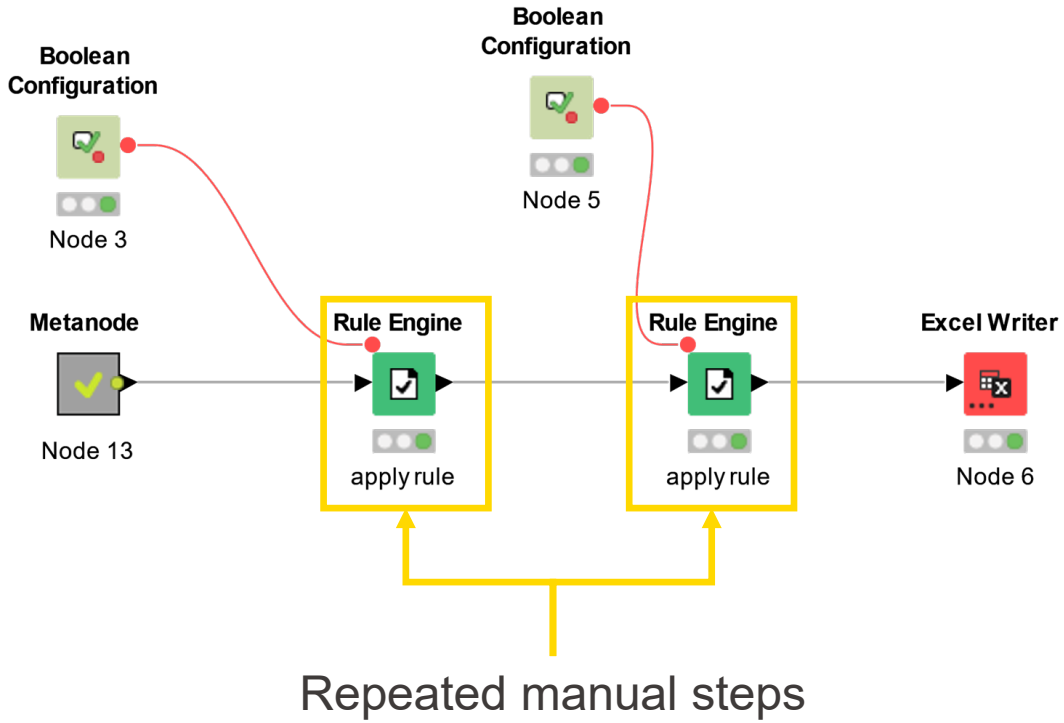
Hard to Reuse Workflow



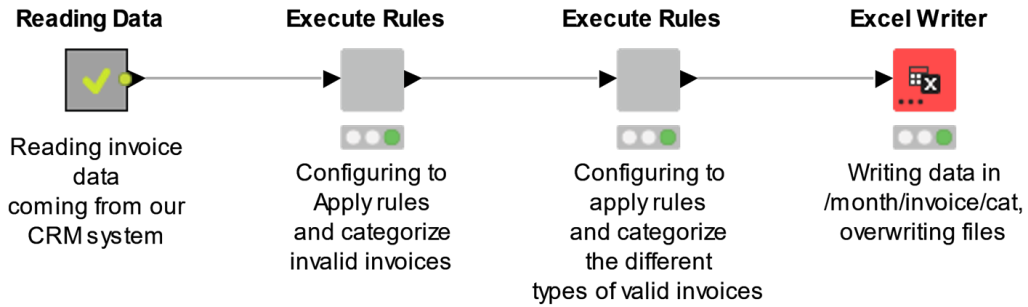
Hard to Reuse Workflow



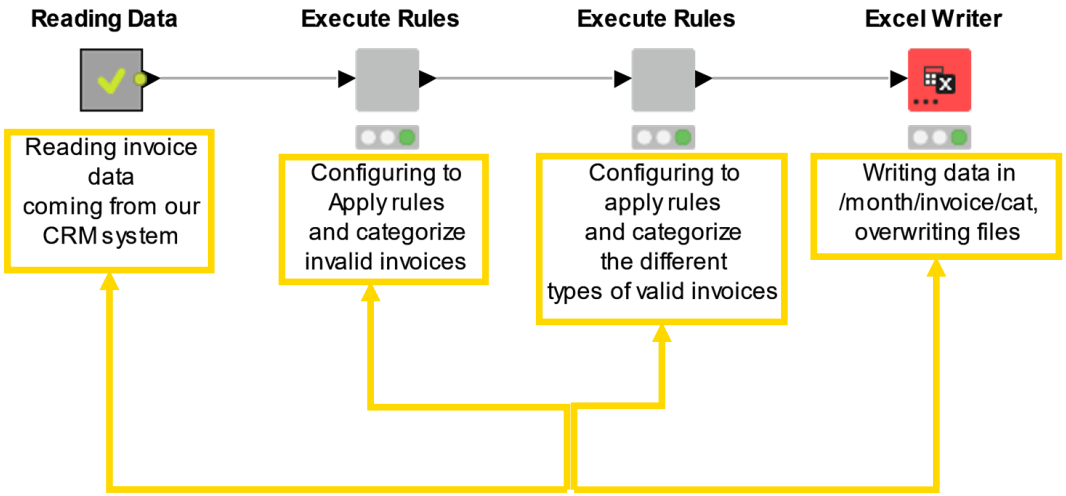
Hard to Reuse Workflow



Easy to Reuse Workflow

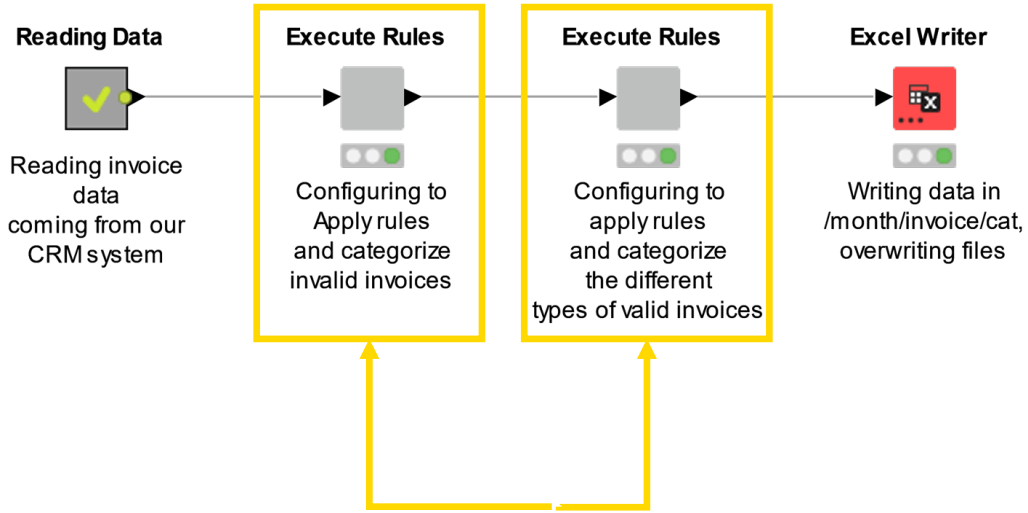


Easy to Reuse Workflow



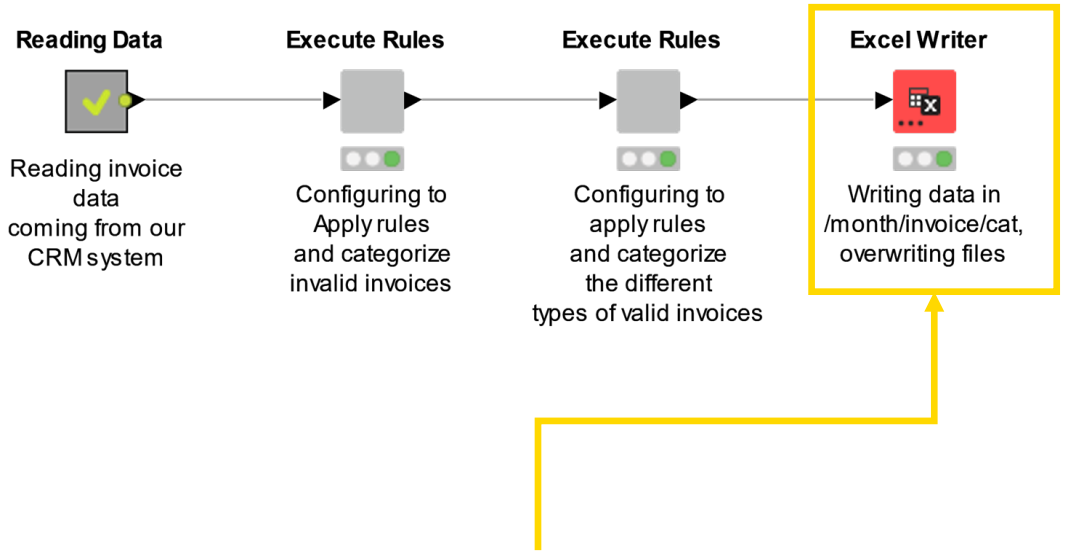
Nicely documented processes

Easy to Reuse Workflow



Components to make repeatable steps reusable

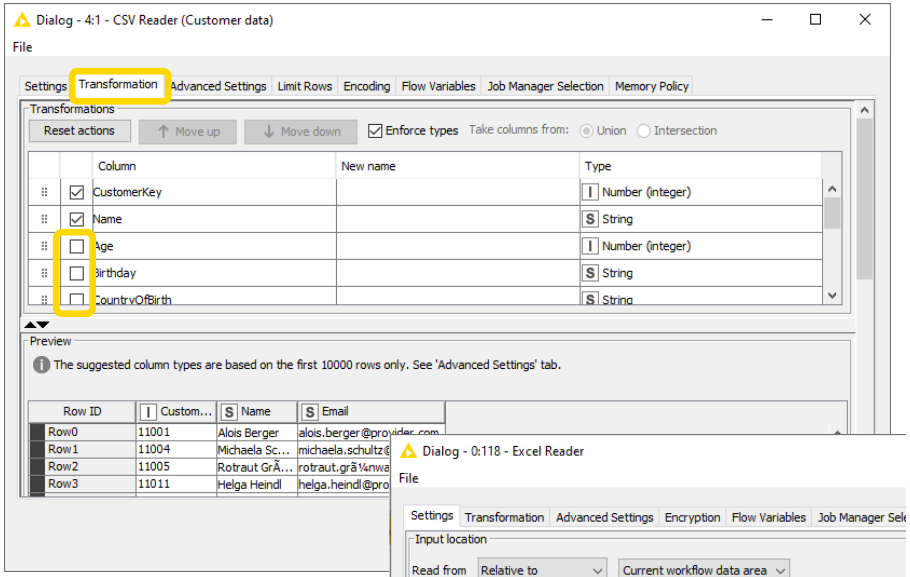
Easy to Reuse Workflow

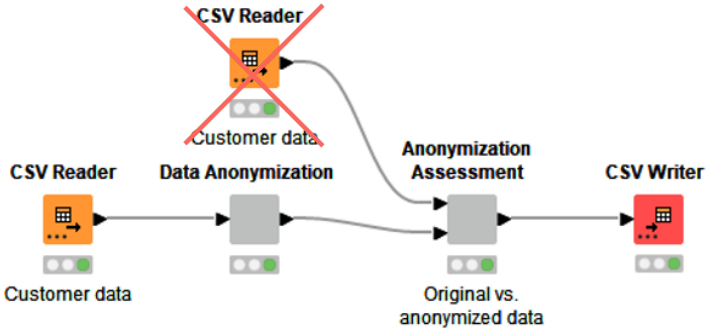


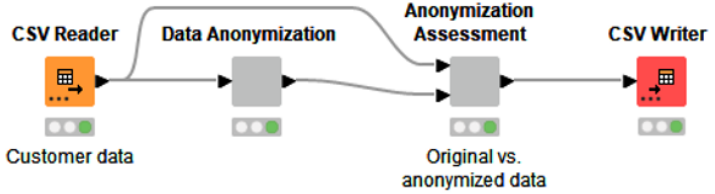
Efficiency

Efficient Reading of Files

- Remove redundant and unused columns in the “transformation” tab of a reader node
- Avoid reading the same file multiple times
- Read multiple files with the same structure with one Reader node using the “Files in folder” option

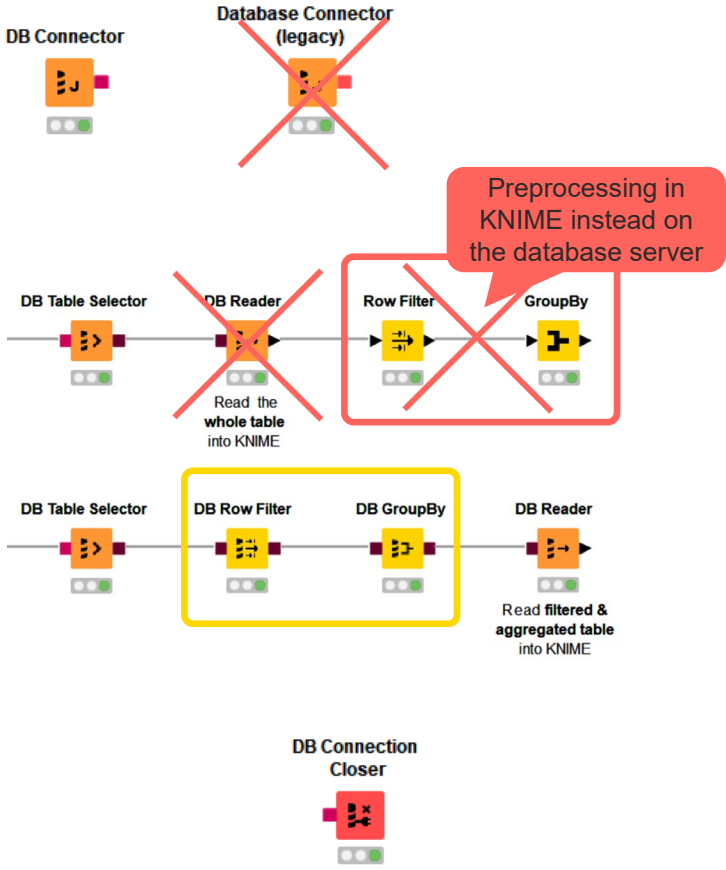




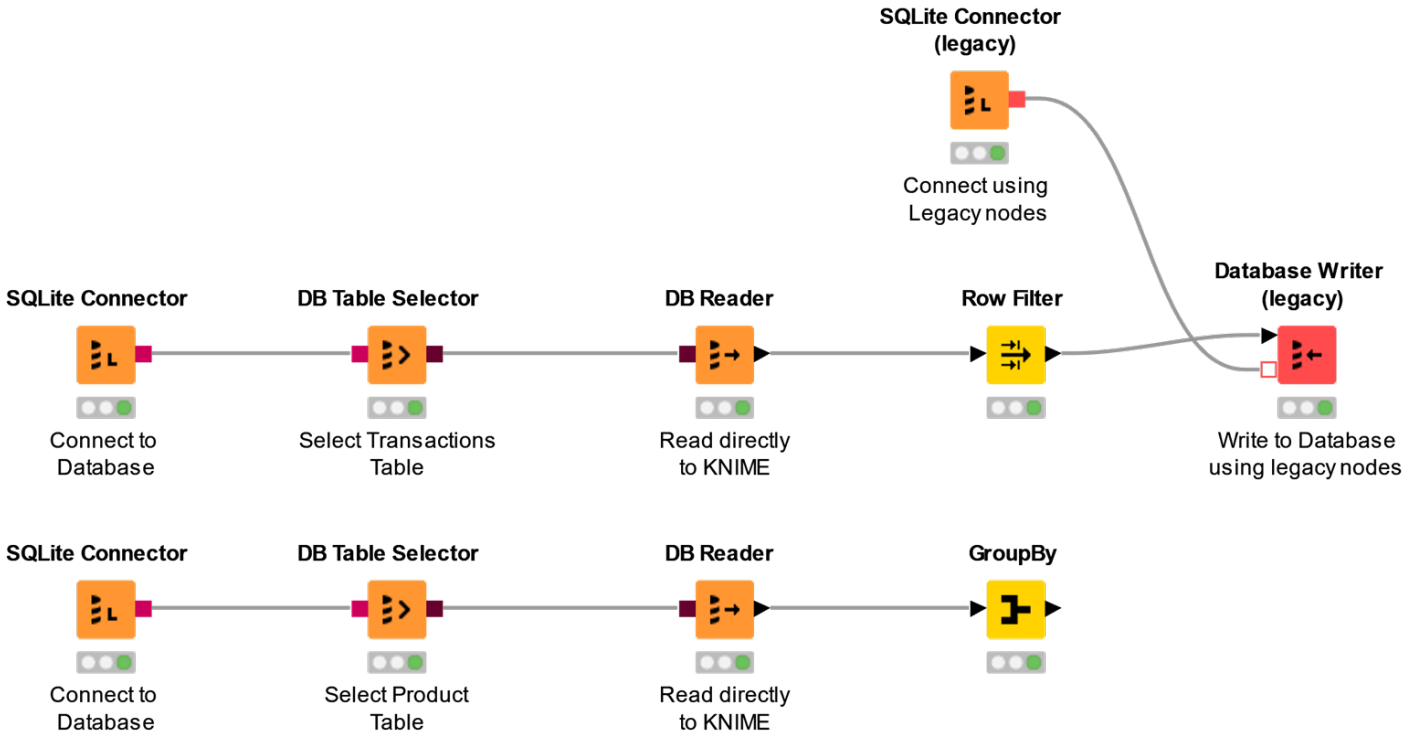


Efficient Usage of Databases

- Make sure to use the new Database framework instead of the legacy/deprecated version
  - “DB nodes” are new, “Database nodes” are legacy and since Dec. 2022 deprecated
- Avoid using the DB connection node multiple times if connecting to the same database
  - Unless you need to open up parallel connections to speed up execution of parallel branches
- Push processing to database server when possible – don’t do it locally
- Always use the DB Connection Closer node at the end of your DB processing

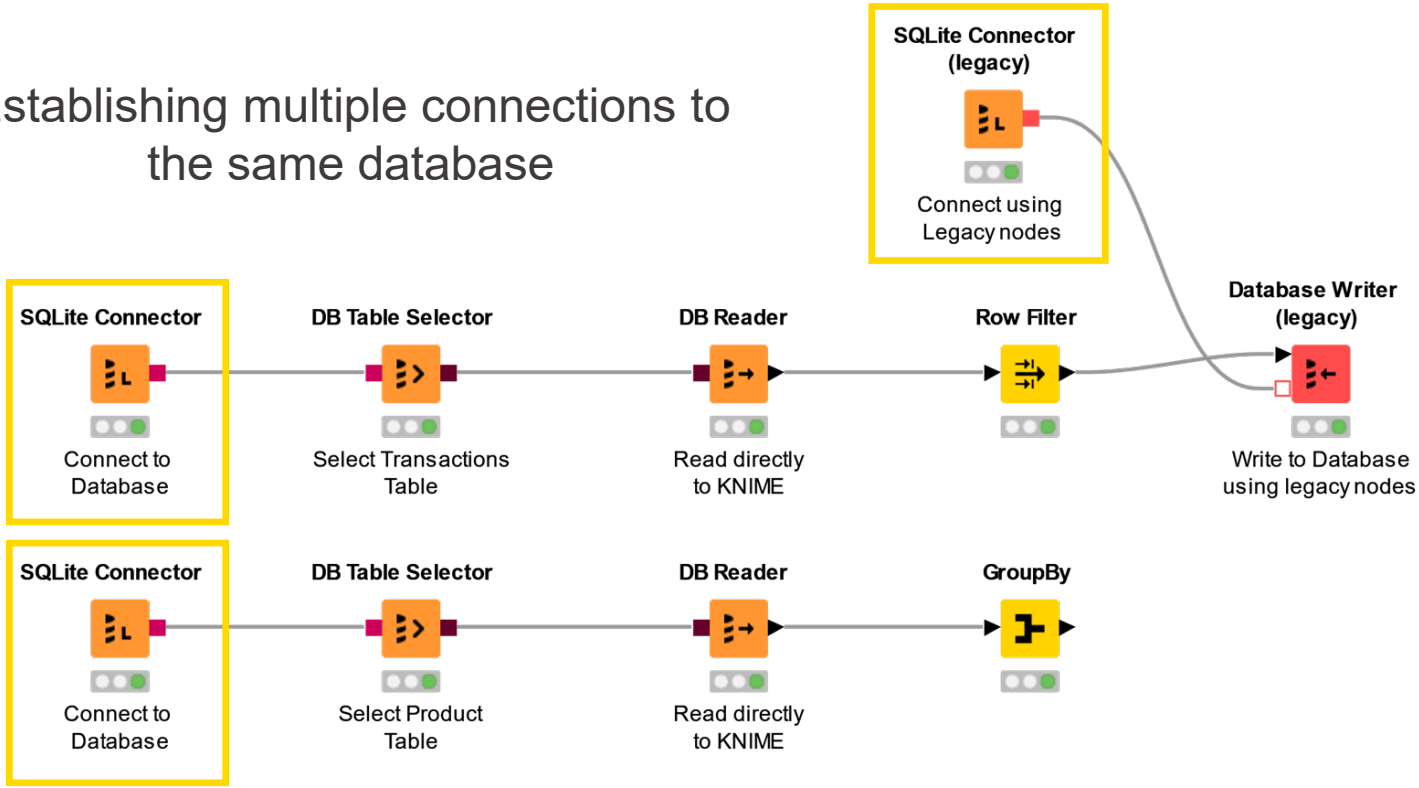


Inefficient Database Workflow



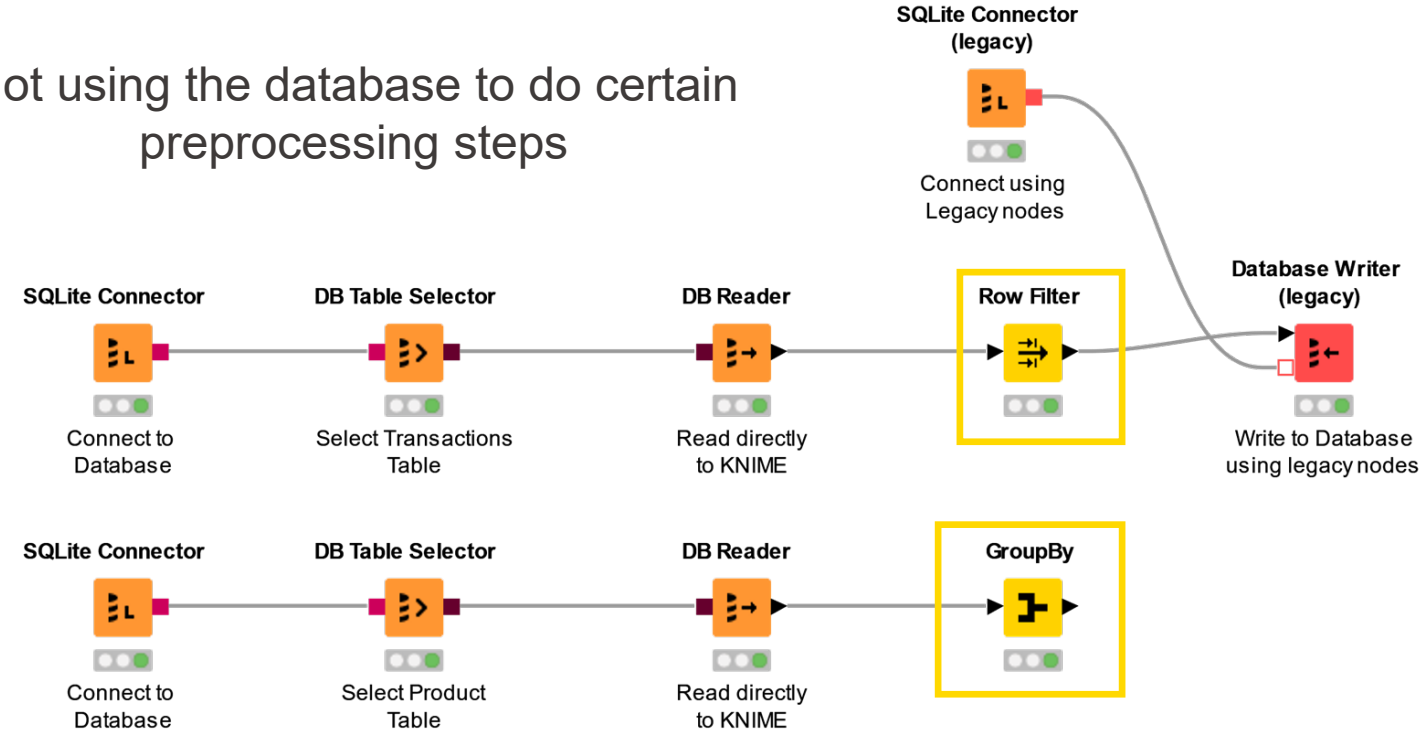
Inefficient Database Workflow

Establishing multiple connections to the same database



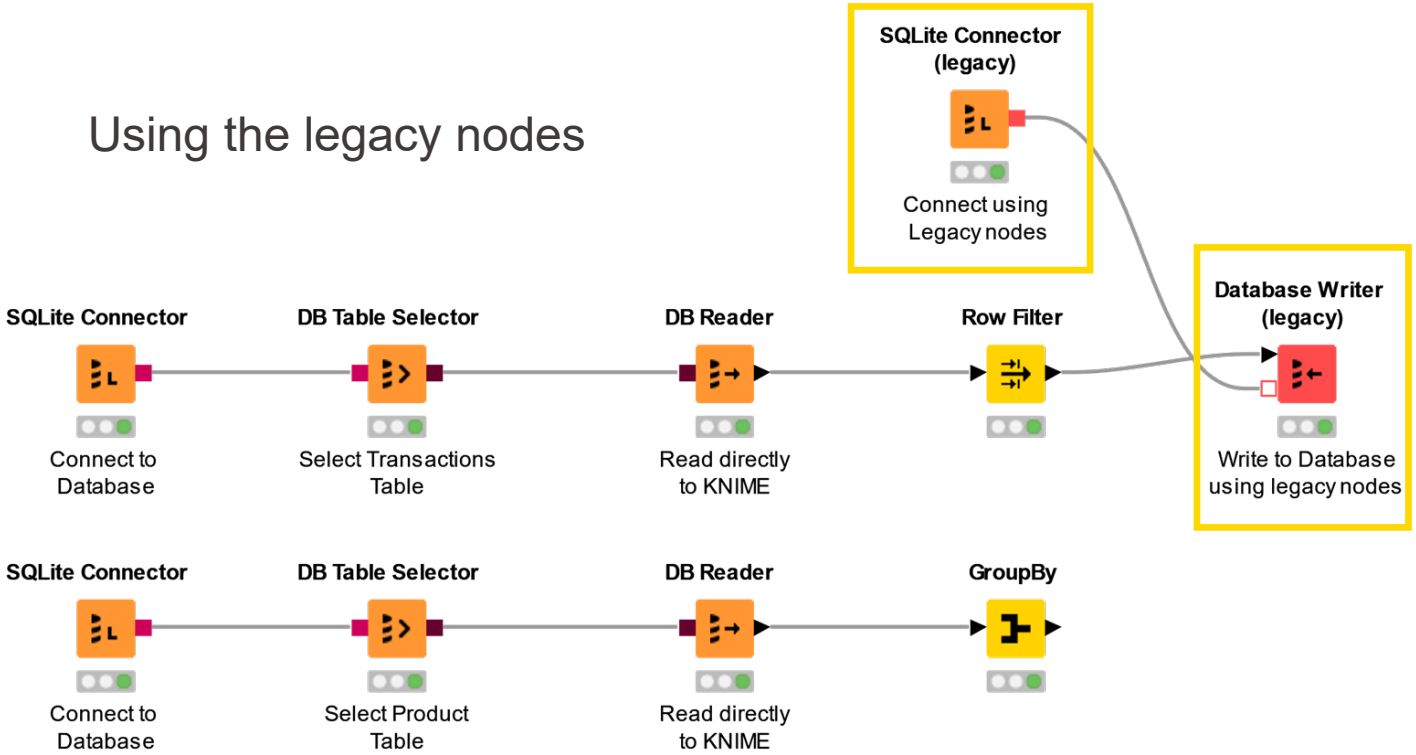
Inefficient Database Workflow

Not using the database to do certain preprocessing steps

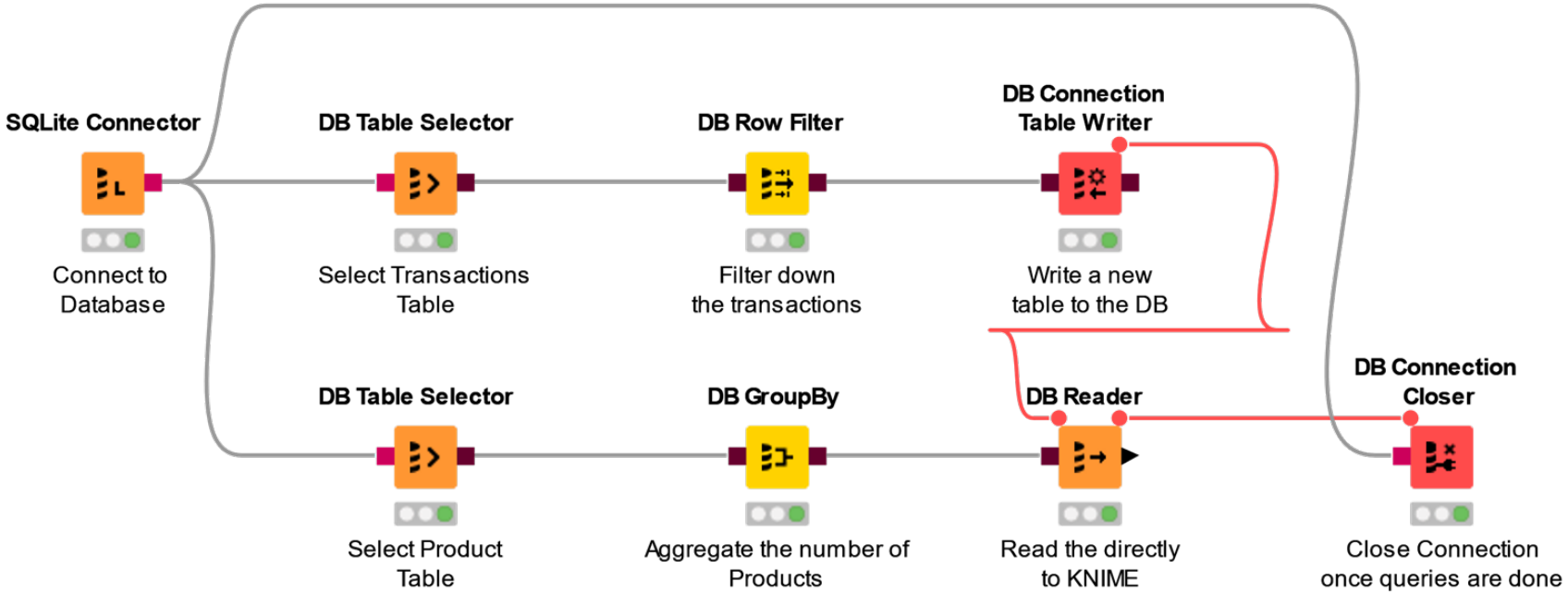


Inefficient Database Workflow

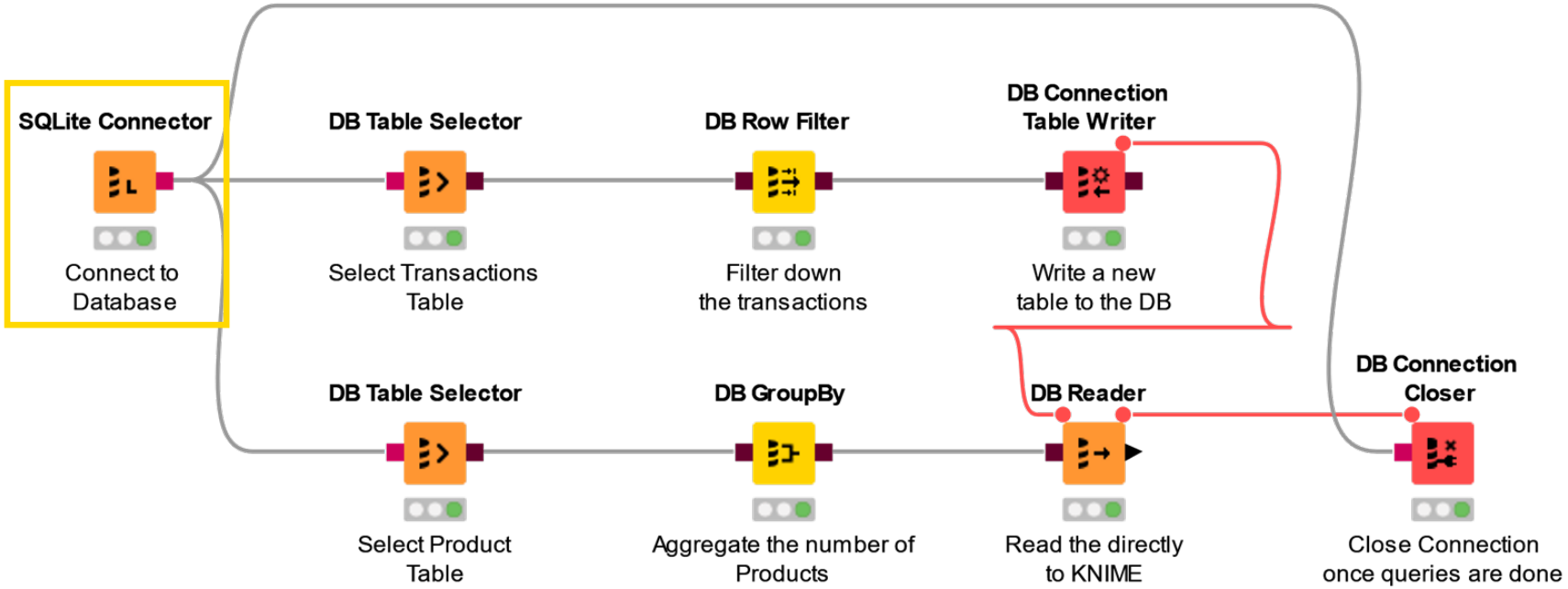
Using the legacy nodes



Efficient Database Workflow

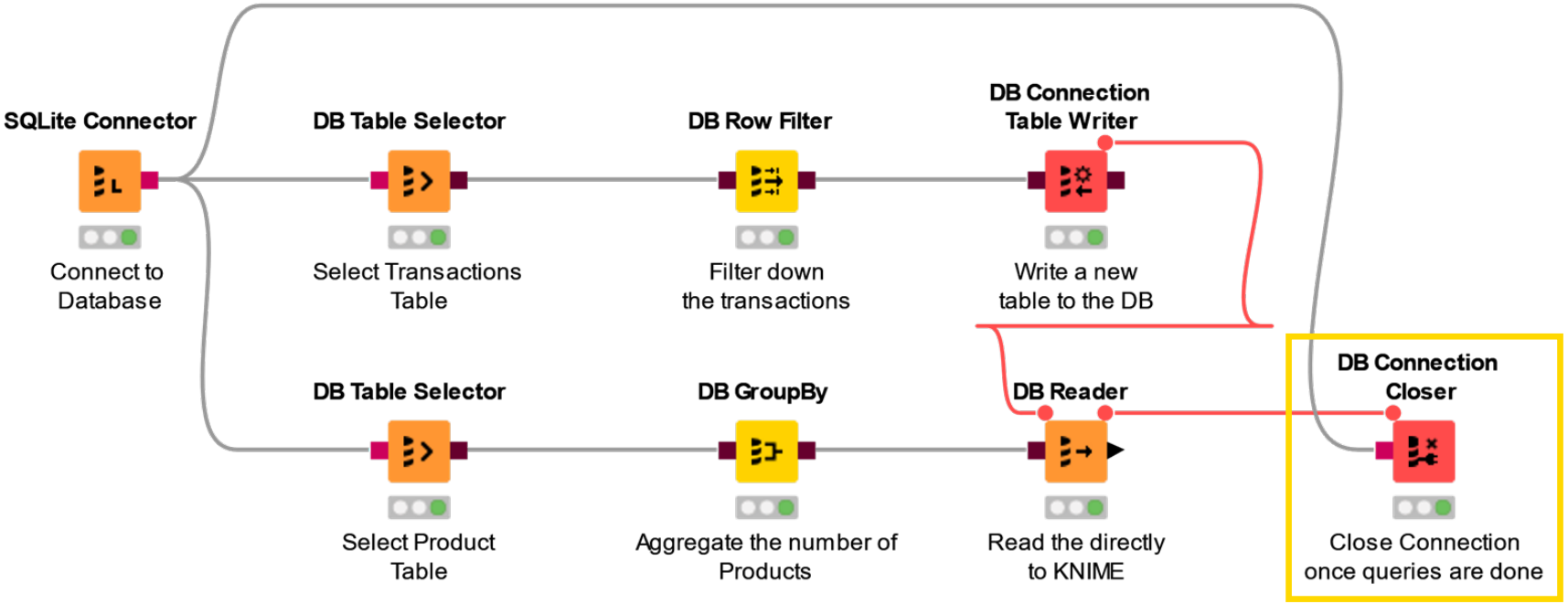


Efficient Database Workflow



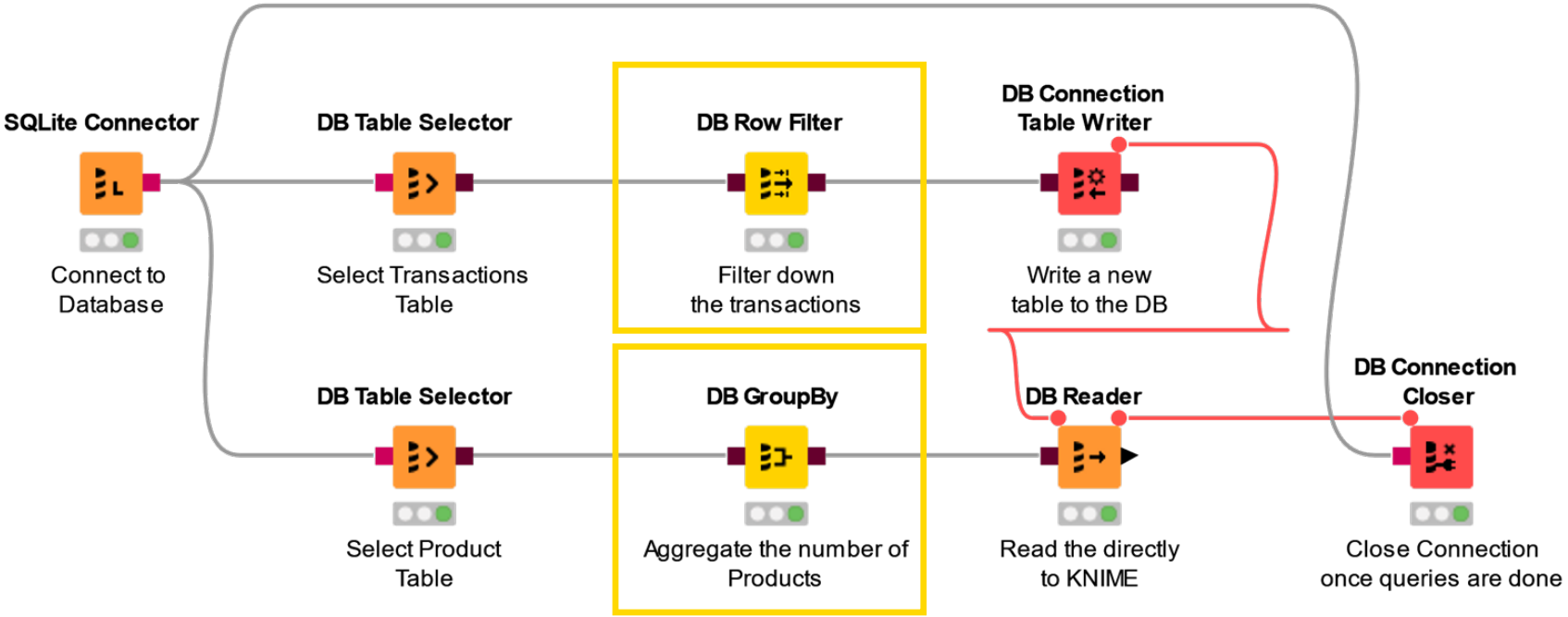
Connecting to the database only the necessary number of times*

Efficient Database Workflow



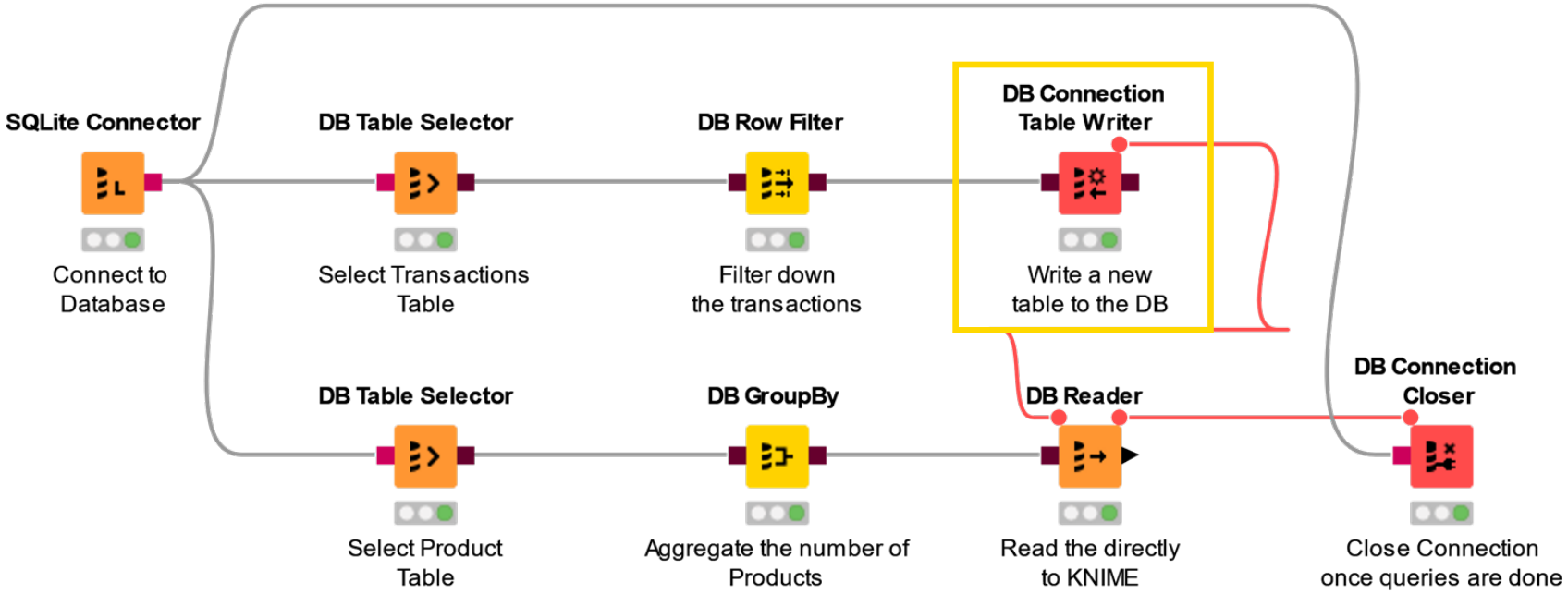
Closing the connection once all database processing is done

Efficient Database Workflow



Doing some preprocessing in the database when possible

Efficient Database Workflow

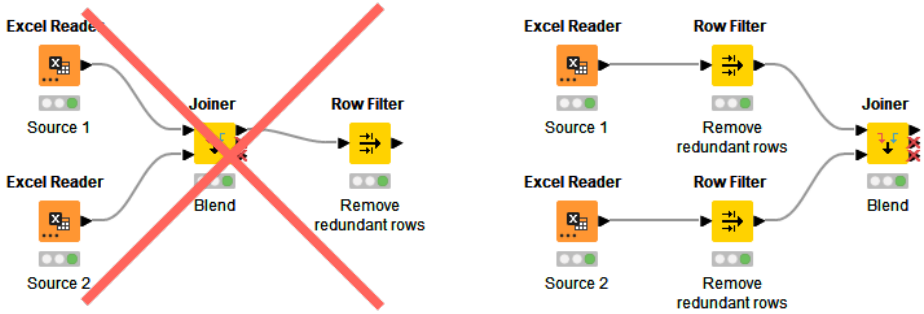


Write the data directly in the database

Efficient Workflow Design

- Not every manipulation operation in your data will be equally expensive
  - Joining two tables with a Joiner node will be more expensive than replacing a few rows with a Cell Replacer node
  - Only use loops when absolutely needed – use Multi Column nodes instead

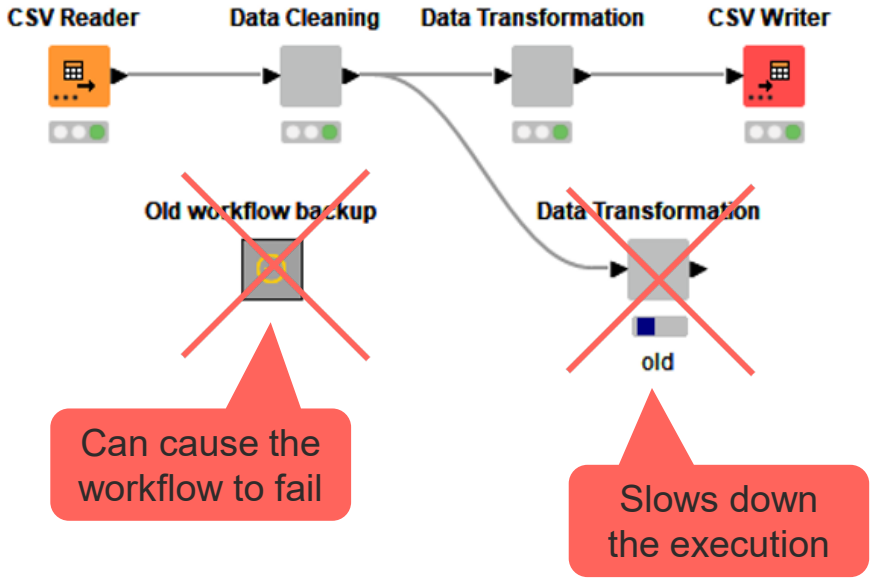
- Not every data type takes up the same amount of memory:
  - Strings occupy more space in memory than integers
  - Consider removing columns with constant values if they are not needed
- Remove redundant data early
  - Filter redundant rows or columns before dragging them through the data pipeline and other data operation, e.g., before joining



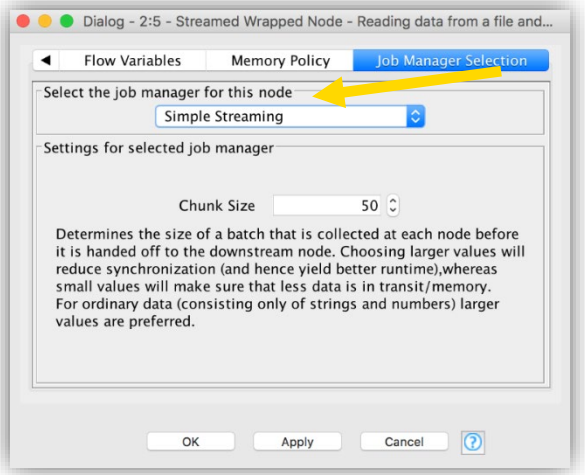
Efficient Workflow Design

Absolutely avoid:

- Disconnected nodes/components, especially if that workflow will run on a KNIME Business Hub or will be called by another workflow
- Workflow branches that are not actively needed
- Storing large unused files inside of the workflow
  - It will cause the workflow to take a long time to load/save

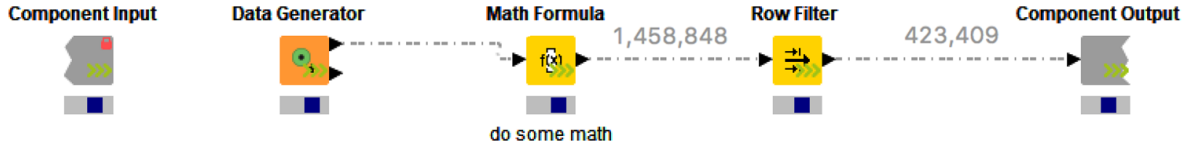


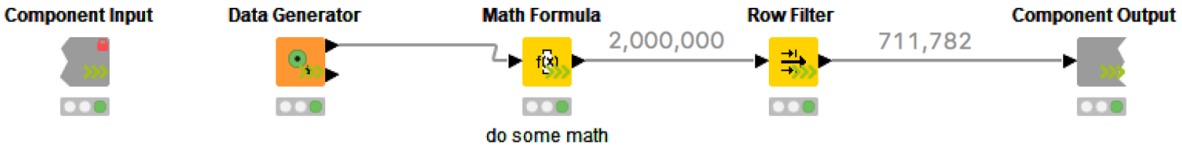
Pro Tip: Streaming



Streamed Component - Reading data from a file and process

Sub Workflow as Component.
To open it:
- right click > Component > Open
- Ctrl + Double Click

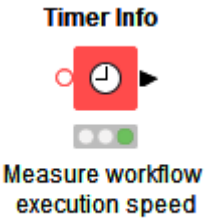




Pro Tip: Timer Node

You can only make your workflow better if you know what is taking time

- Make use of the timer node to measure your workflow:
  - Find out which nodes take the longest to execute
  - Try to figure it out what is KNIME-related and what could be related to other variables (i.e. slow network connection)
  - Measure how much you have improved your execution time – keep track of success



Output table - 4:22 - Timer Info (Measure workflow)

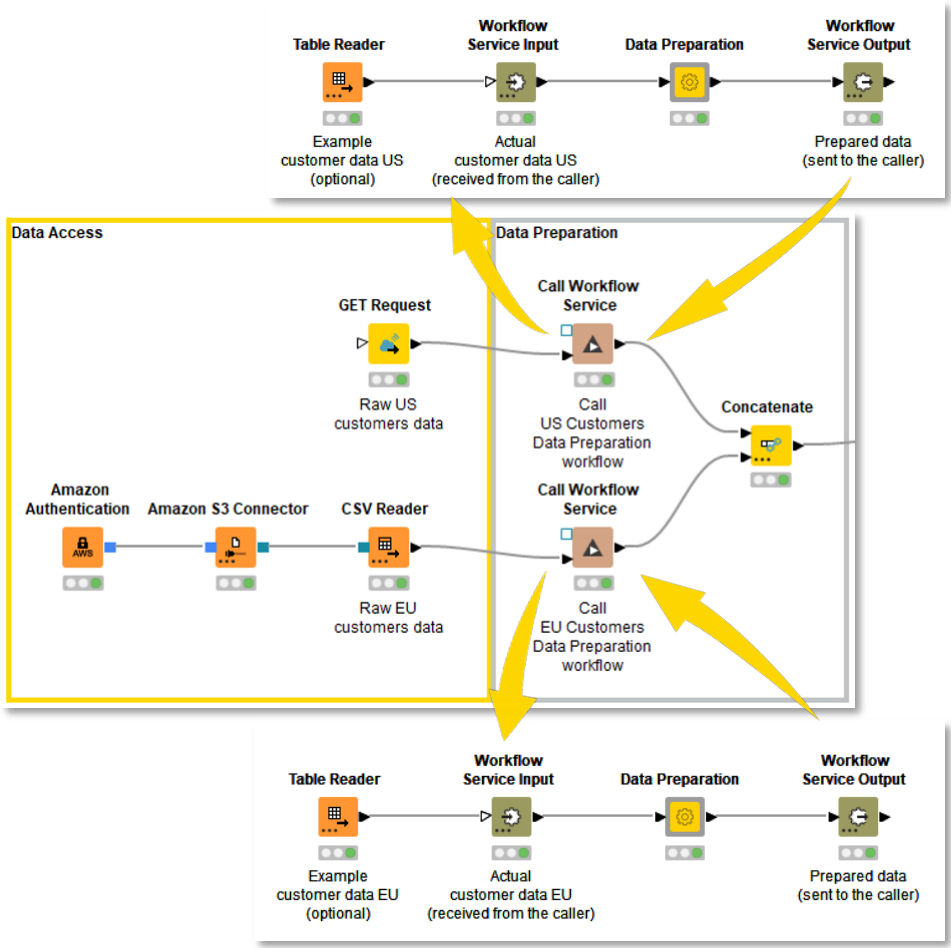
File Edit Hilite Navigation View

Table "default" - Rows: 8 | Spec - Columns: 8 | Properties | Flow Variables

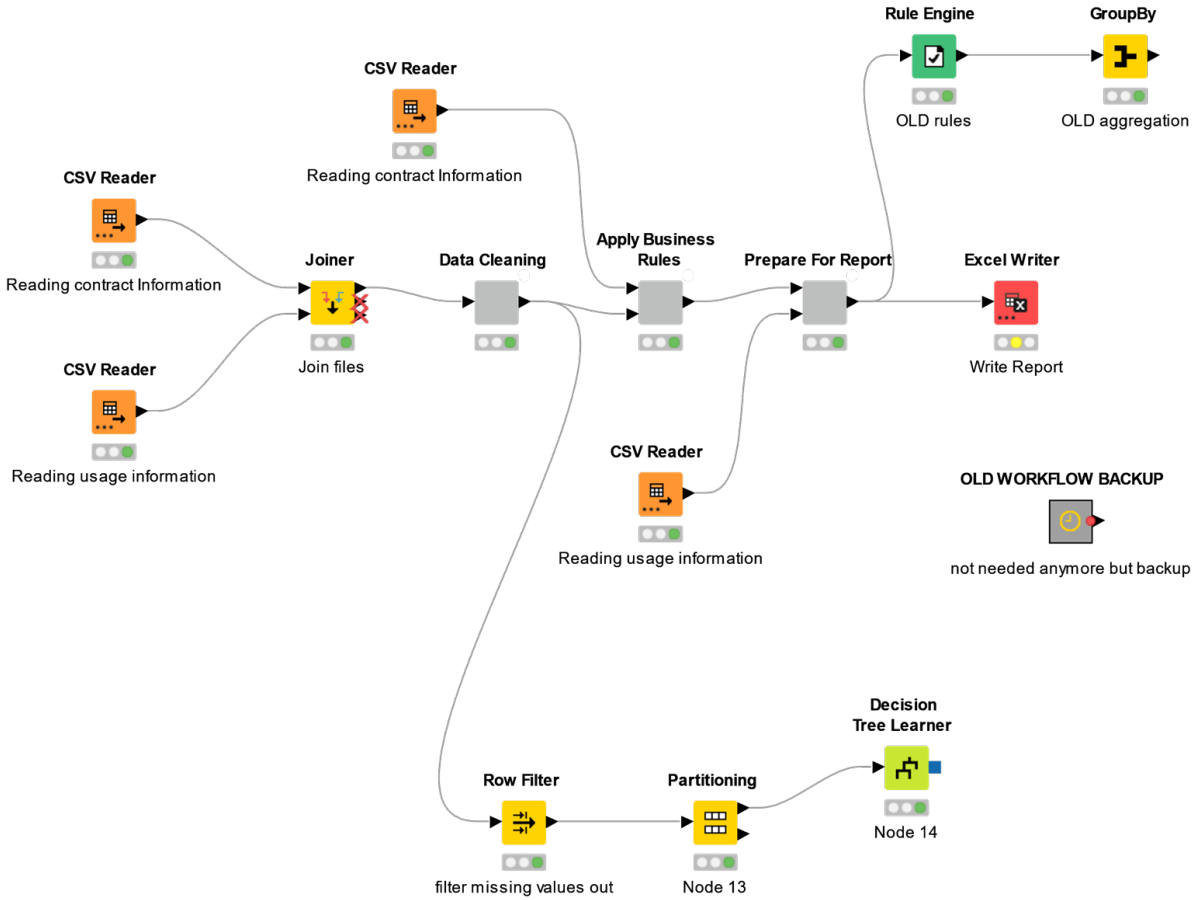
Row ID	Name	Execution Time	Execution Time since last Reset	Execution Time since Start	Nr of E...	Nr of E...	NodeID	Classname
Node 1	CSV Reader	892	892	892	1	1	4:1	org.knime.filehandling.core.node.table.reader.TableReaderNodeModel
Node 2	Excel Reader	832	832	832	1	1	4:2	org.knime.ext.poi3.node.io.filehandling.excel.reader.ExcelTableRea...
Node 3	Cell Replacer	17	17	17	1	1	4:3	org.knime.base.node.preproc.cellreplace.CellReplacerNodeModel
Node 18	SQLite Conn...	274	274	274	1	1	4:18	org.knime.database.extension.sqlite.node.connector.SQLiteDBConn...
Node 19	DB Table Sel...	31	31	31	1	1	4:19	org.knime.database.node.utility.tableselector.DBTableSelectNodeMo...
Node 20	DB Reader	107	107	107	1	1	4:20	org.knime.database.node.io.reader.DBReadNodeModel
Node 21	Joiner	91	91	91	1	1	4:21	org.knime.base.node.preproc.joiner3.Joiner3NodeModel
Node 22	Timer Info	?	0	0	0	0	4:22	org.knime.base.node.util.timerinfo.TimerinfoNodeModel

Pro Tip: Divide and Conquer

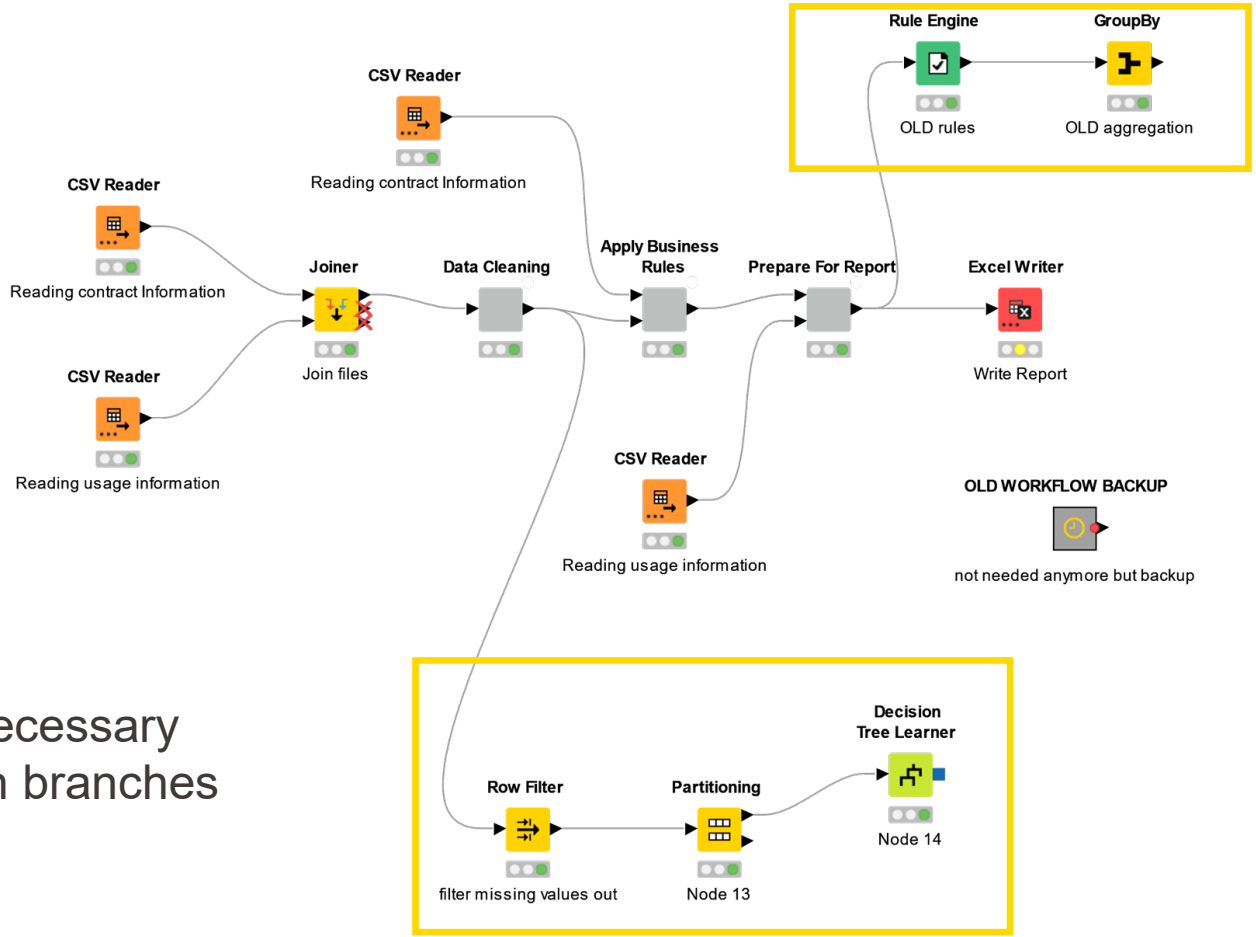
- Create modular workflows
- Control their input, execution, and output from the master workflow
- Benefits over one big workflow
  - Each workflow performs a particular task
  - Multiple, smaller workflows can be faster than one big workflow
  - Easier to maintain
  - Easier to test
  - Faster to load



Inefficient Workflow Design

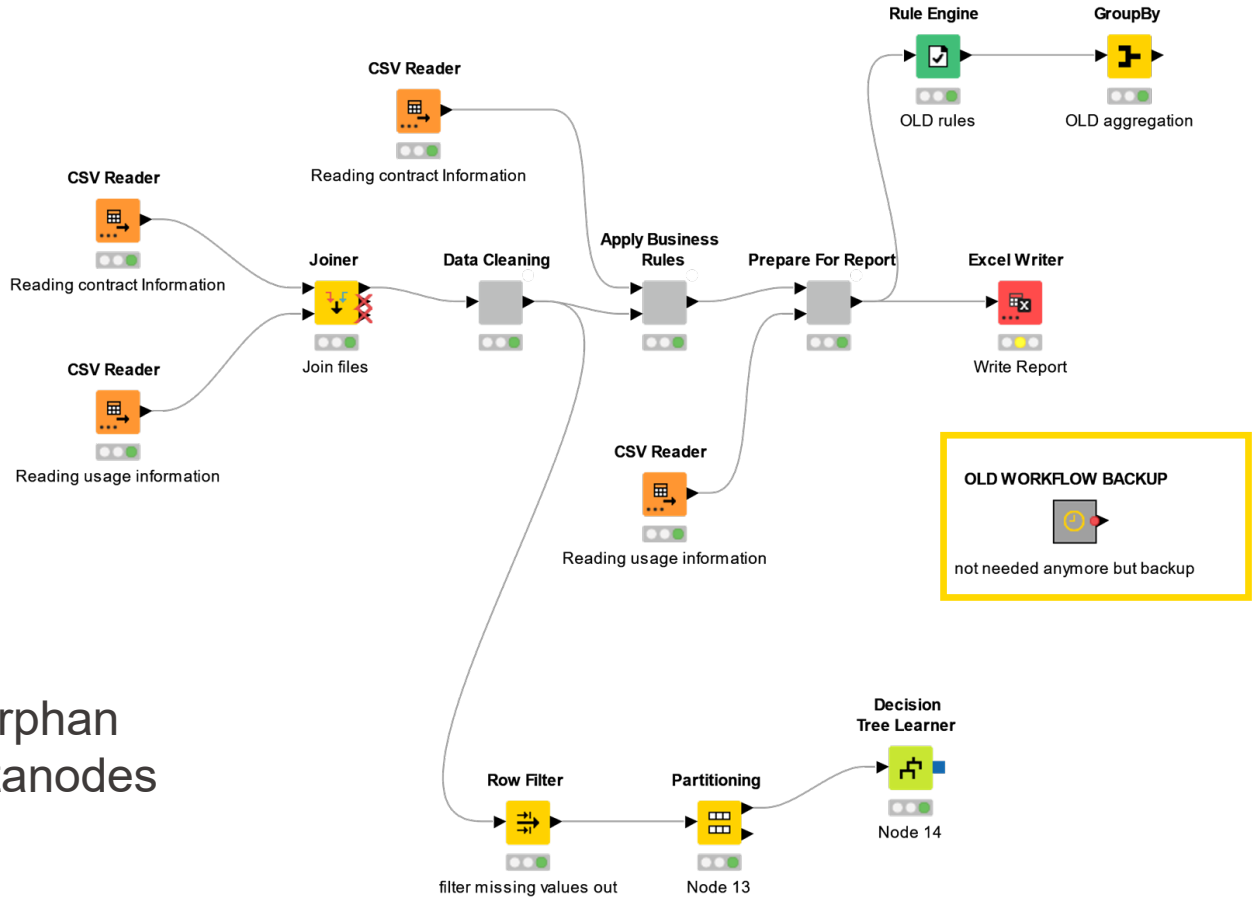


Inefficient Workflow Design

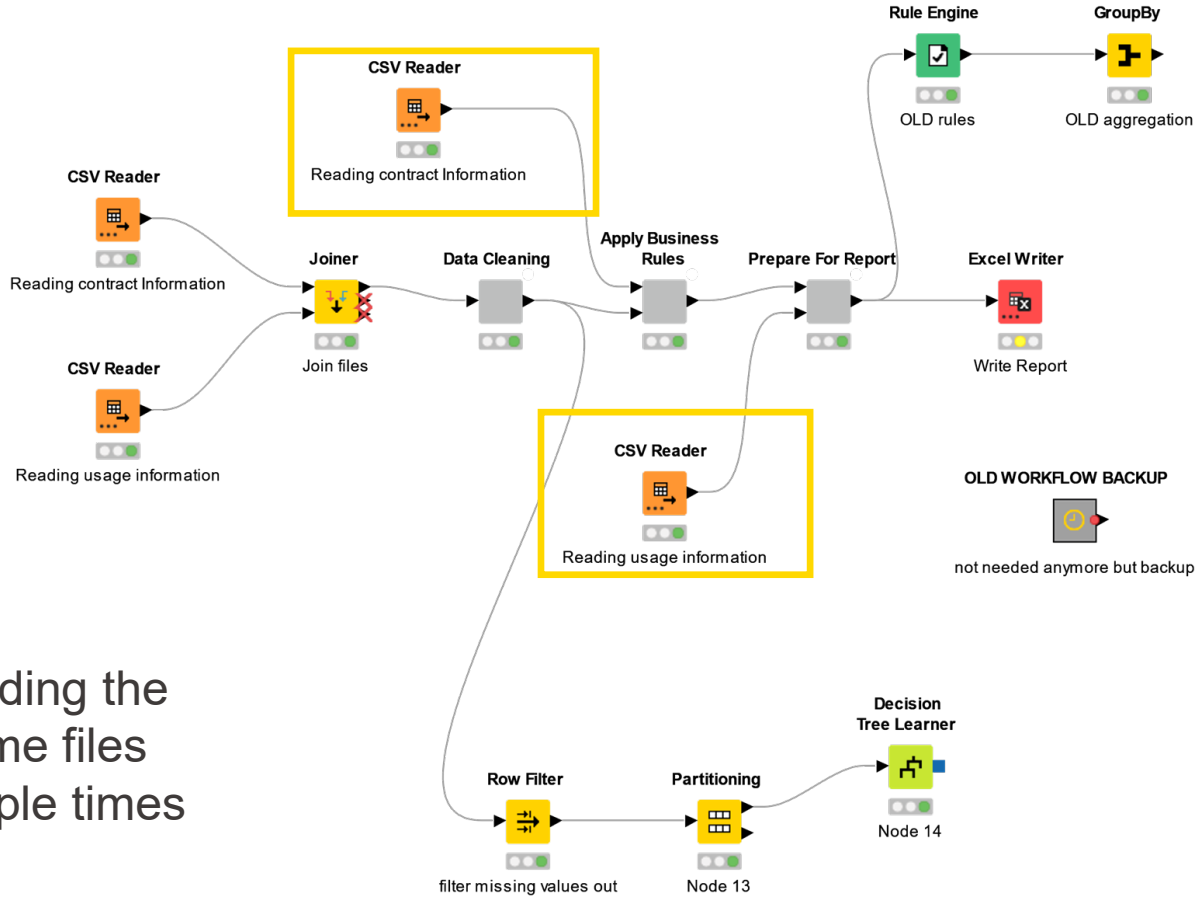


Unnecessary orphan branches

Inefficient Workflow Design

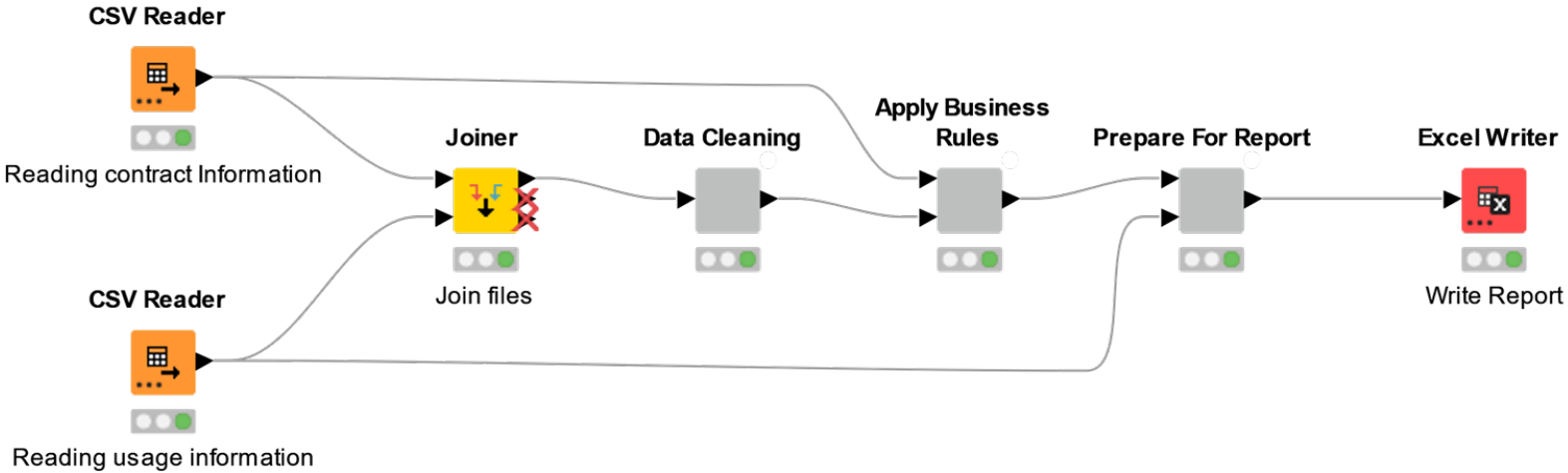


Inefficient Workflow Design

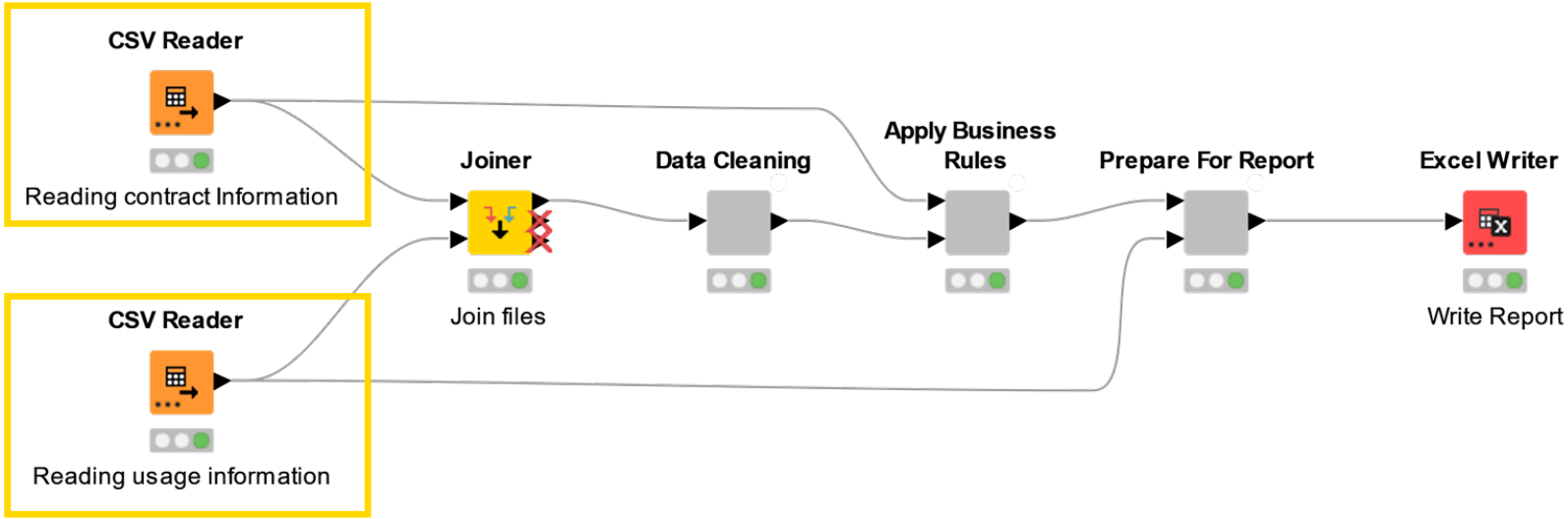


Reading the same files multiple times

Efficient Workflow Design



Efficient Workflow Design



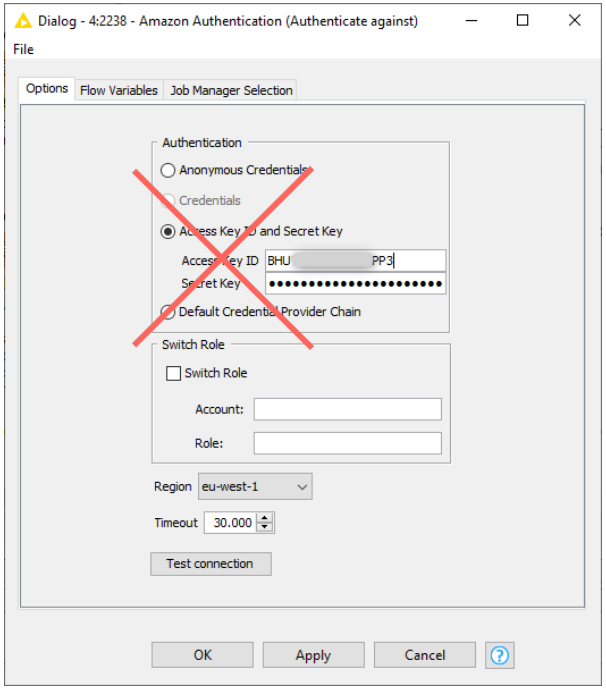
Files are read only once, but connections are reused

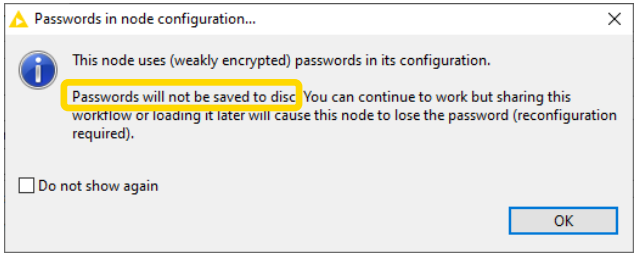
Security

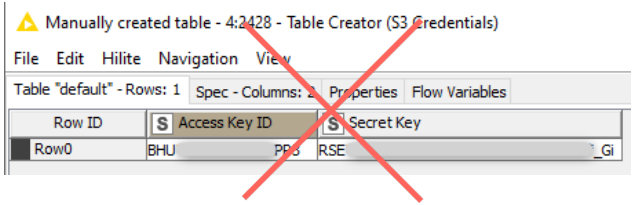
Security Considerations

- No passwords in plaintext anywhere
- Do not hardcode your credentials in your nodes
- To always avoid storing passwords as part of the workflow on disc:
  - Specify permanently in the knime.ini in the root of the KNIME installation:

    -Dknime.settings.passwords.forbidden=true
- Do not save confidential data inside your workflow
  - Reset your workflow before sharing it (unless there is a specific reason not to)

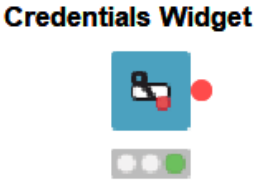


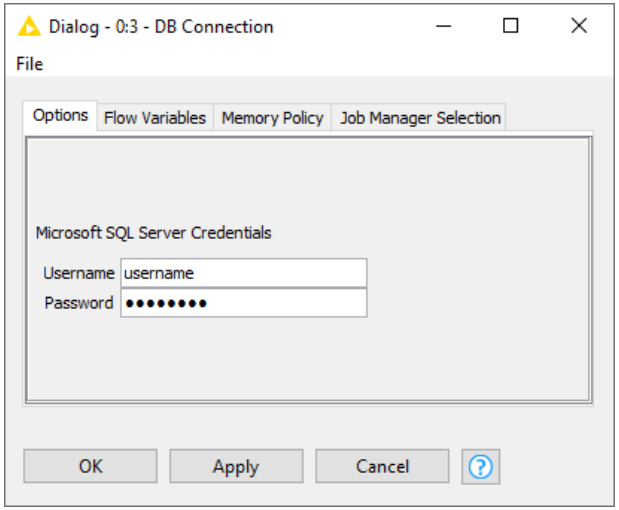


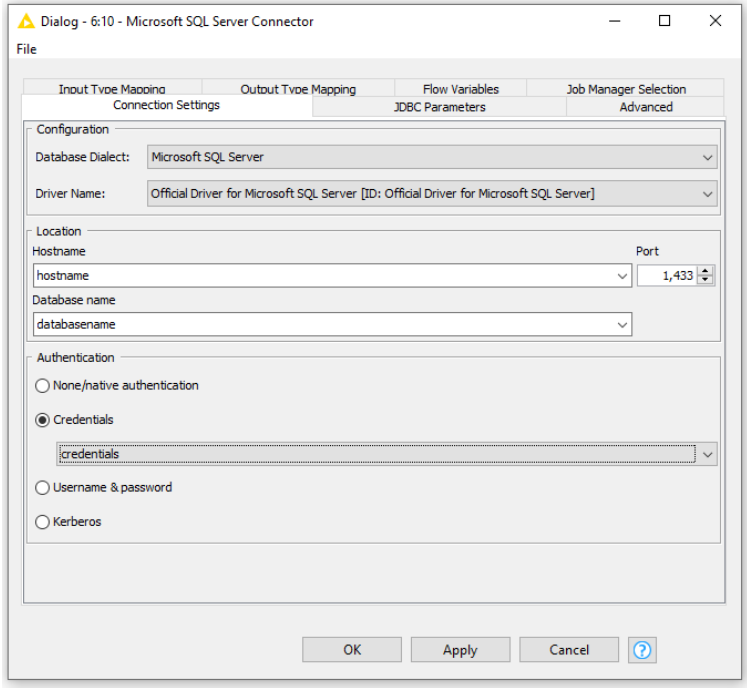


Handling Credentials

- Avoid saving credentials to the node
- Use credential widgets/configuration nodes to let the user specify credentials at runtime

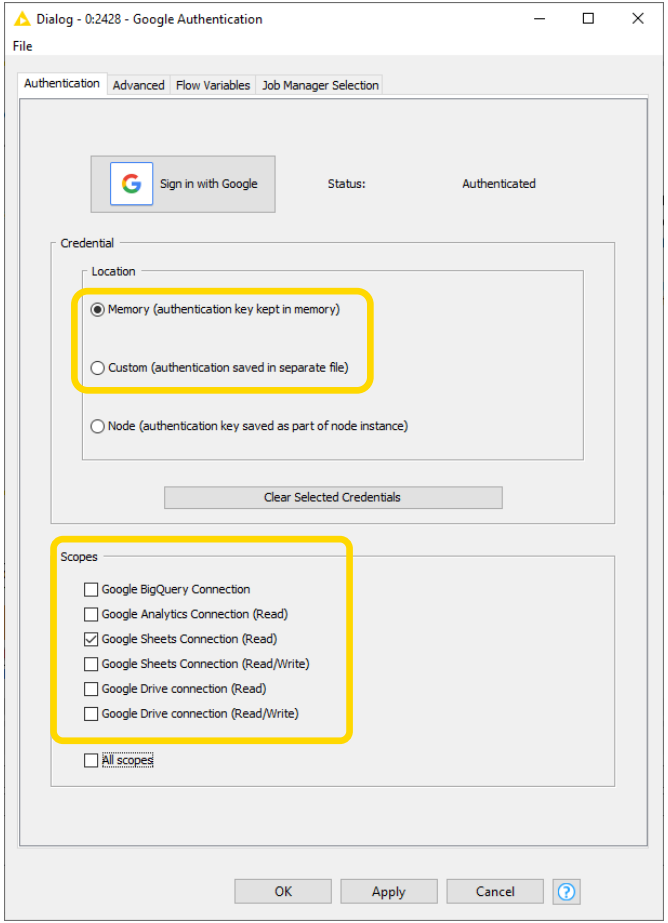




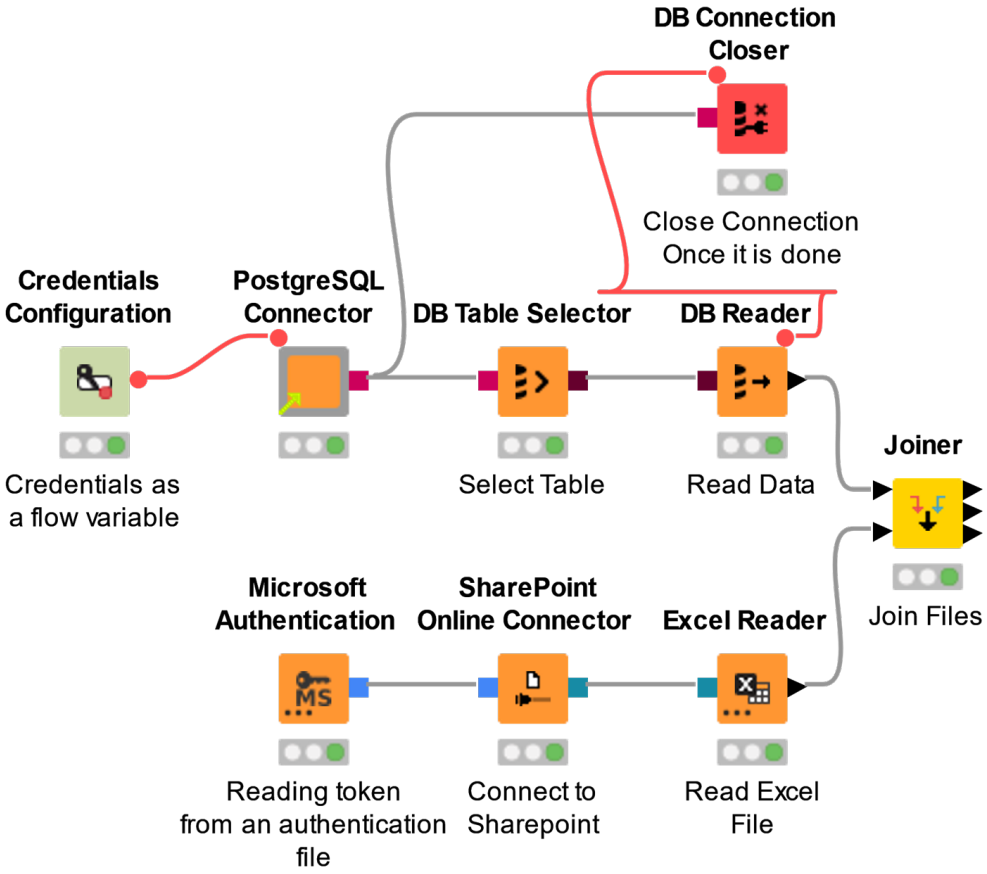


Handling Cloud Credentials

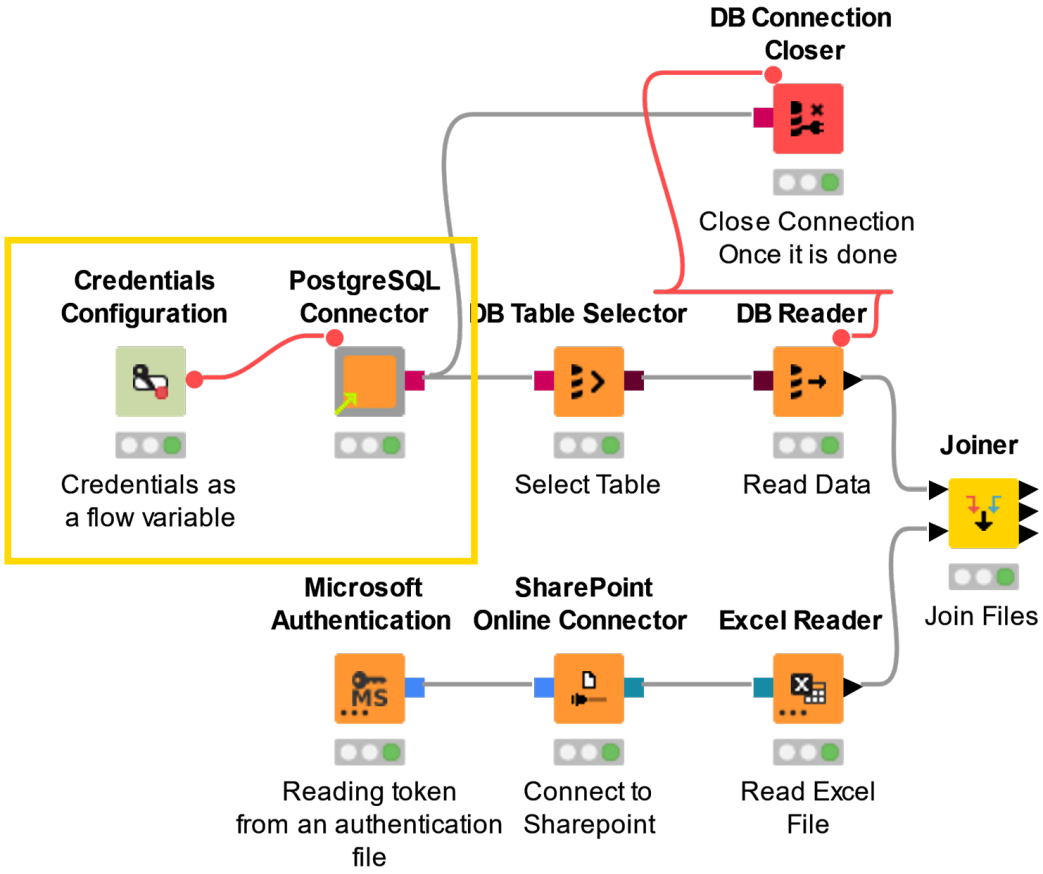
- Avoid saving the authentication credentials to the node
- If you will use the node only once, pick to keep the authentication credentials in memory
- If you want to use the node more than once, make use of authentication files
- When possible, limit the available scopes



Best Practice for Handling Credentials

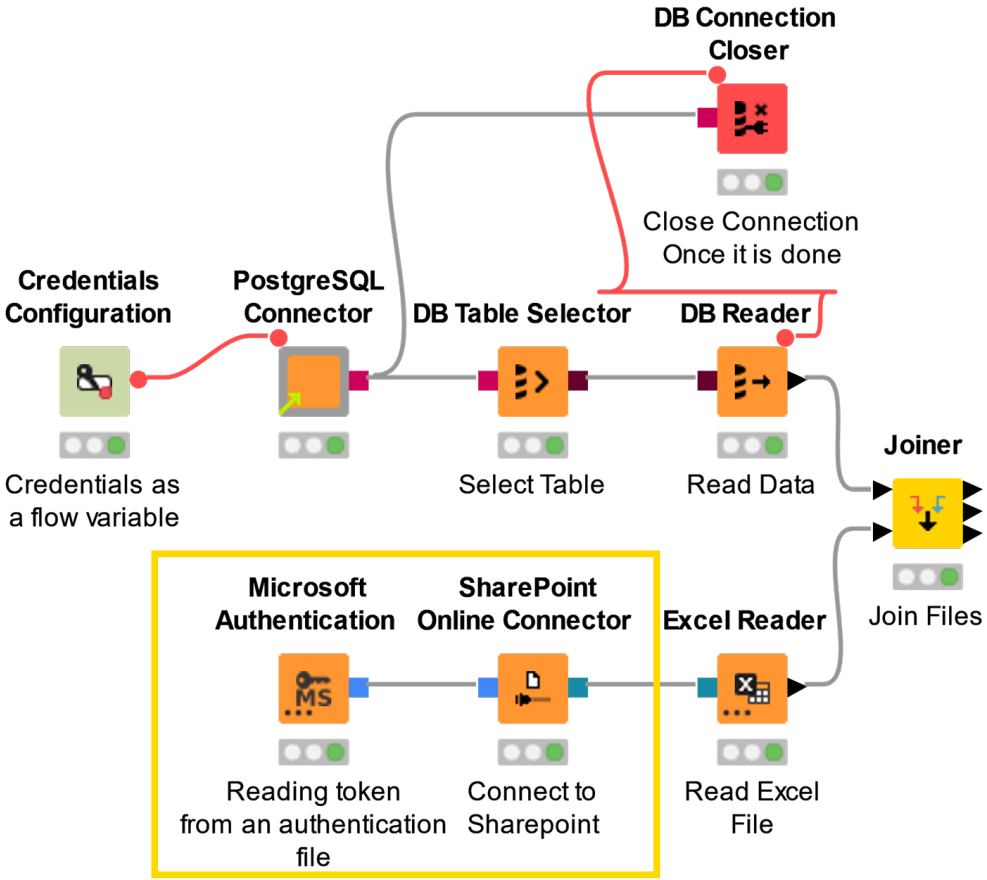


Best Practice for Handling Credentials



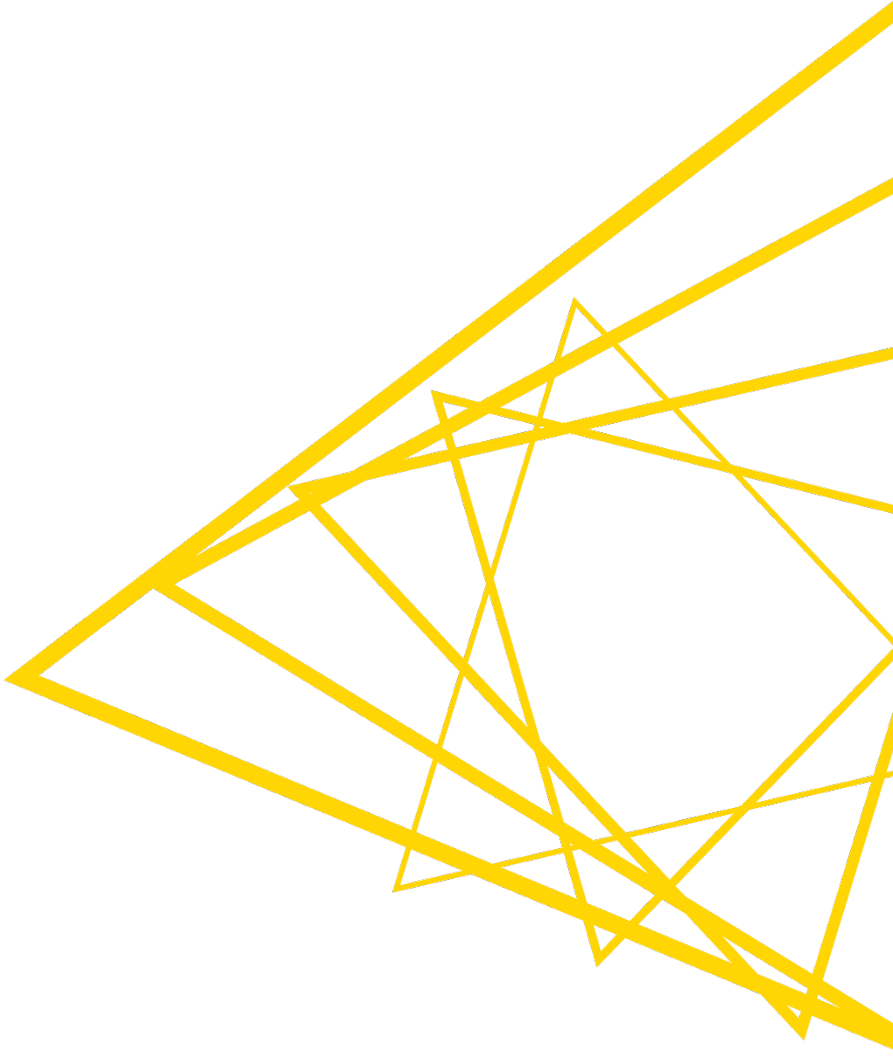
Using credentials configuration node to setup credentials

Best Practice for Handling Credentials

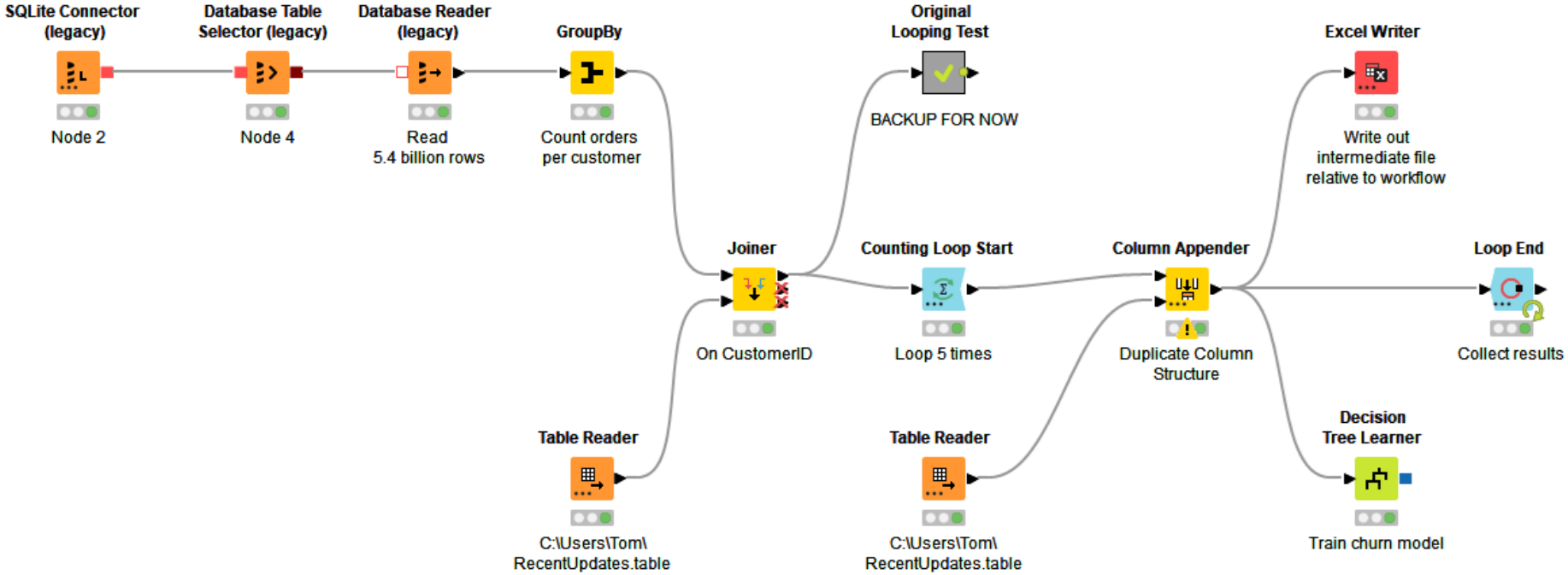


Connecting to cloud resources with an authentication file

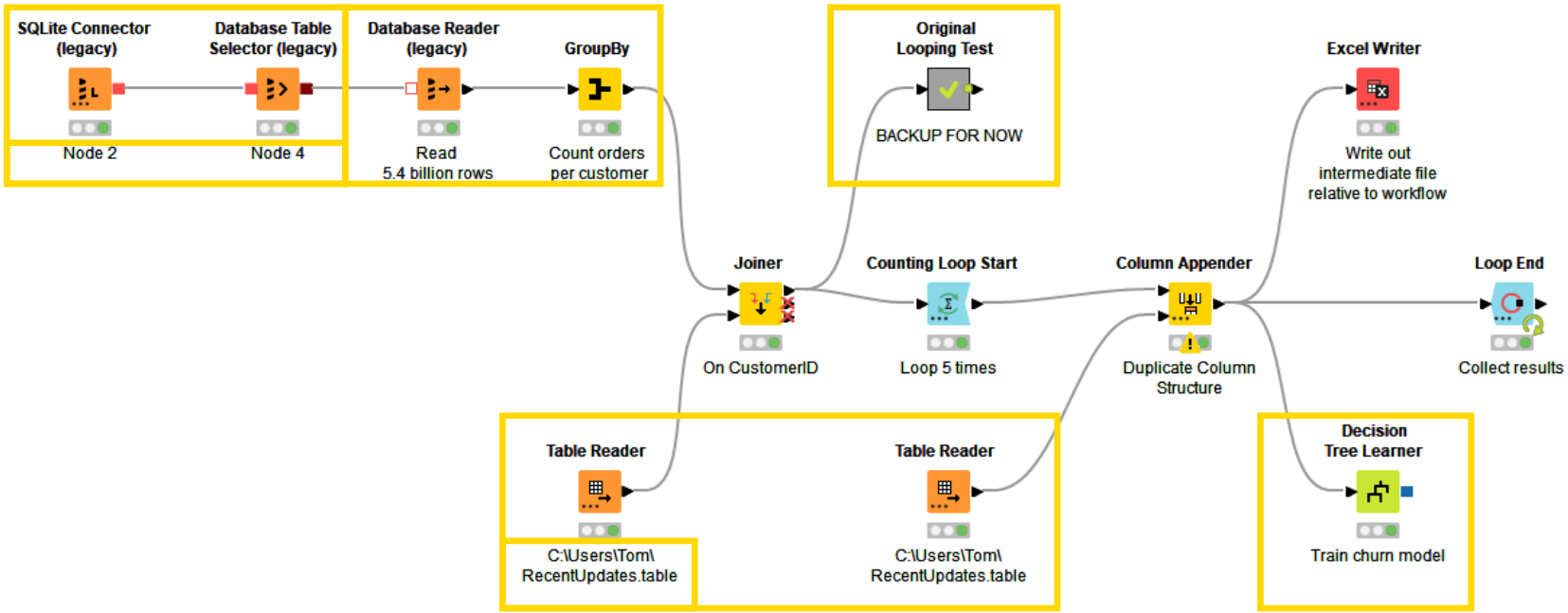
Quiz: Find the mistakes!



What’s wrong with this picture?



What's wrong with this picture?



- Legacy database nodes
- Poor node annotations
- Not pushing down processing to database
- Reading flat files twice

- Local file path on Table Reader
- Orphaned metanode for backup
- Orphaned learner node (inside loop!)
- No useful components or metanodes

The Pillars of a Good Workflow

Reusable

Secure

Efficient

Additional Resources

- Blog posts
  - https://www.knime.com/blog/optimizing-knime-workflows-for-performance
  - https://www.knime.com/blog/declutter-four-tips-for-an-efficient-fast-workflow
  - https://www.knime.com/blog/tips-tricks-for-using-knime-analytics-platform-today-bunny-ears-metanodes-co
  - https://www.knime.com/blog/tips-tricks-for-using-knime-analytics-platform-today-ui-nodes

- Cheat sheets
  - https://www.knime.com/cheat-sheets
  - Many to choose from!

Thank you!