Other popular process engines include **Apache Storm**, which utilizes spouts to read data and bolts to perform transformations. By connecting them, you build a processing pipeline. **Apache Flink** and **Samza** are more modern stream and batch processing frameworks that allow you to process unbounded streams. An unbounded stream is data that comes in with no known end – a temperature sensor, for example, is an unbounded stream. It is constantly reporting temperatures. Flink and Samza are excellent choices if you are using Apache Kafka to stream data from a system. You will learn more about Apache Kafka later in this book.

Data pipelines

Combining a transactional database, a programming language, a processing engine, and a data warehouse results in a pipeline. For example, if you select all the records of widget sales from the database, run it through Spark to reduce the data to widgets and counts, then dump the result to the data warehouse, you have a pipeline. But this pipeline is not very useful if you have to execute manually every time you want it to run. Data pipelines need a scheduler to allow them to run at specified intervals. The simplest way to accomplish this is by using **crontab**. Schedule a cron job for your Python file and sit back and watch it run every *X* number of hours.

Managing all the pipelines in crontab becomes difficult fast. How do you keep track of pipelines' successes and failures? How do you know what ran and what didn't? How do you handle backpressure – if one task runs faster than the next, how do you hold data back, so it doesn't overwhelm the task? As your pipelines become more advanced, you will quickly outgrow crontab and will need a better framework.

Apache Airflow

The most popular framework for building data engineering pipelines in Python is **Apache Airflow**. Airflow is a workflow management platform built by Airbnb. Airflow is made up of a web server, a scheduler, a metastore, a queueing system, and executors. You can run Airflow as a single instance, or you can break it up into a cluster with many executor nodes – this is most likely how you would run it in production. Airflow uses **Directed Acyclic Graphs** (**DAGs**).

A DAG is Python code that specifies tasks. A graph is a series of nodes connected by a relationship or dependency. In Airflow, they are directed because they flow in a direction with each task coming after its dependency. Using the preceding example pipeline, the first node would be to execute a SQL statement grabbing all the widget sales. This node would connect downstream to another node, which would aggregate the widgets and counts. Lastly, this node would connect to the final node, which loads the data into the warehouse. The pipeline DAG would look as in the following diagram:

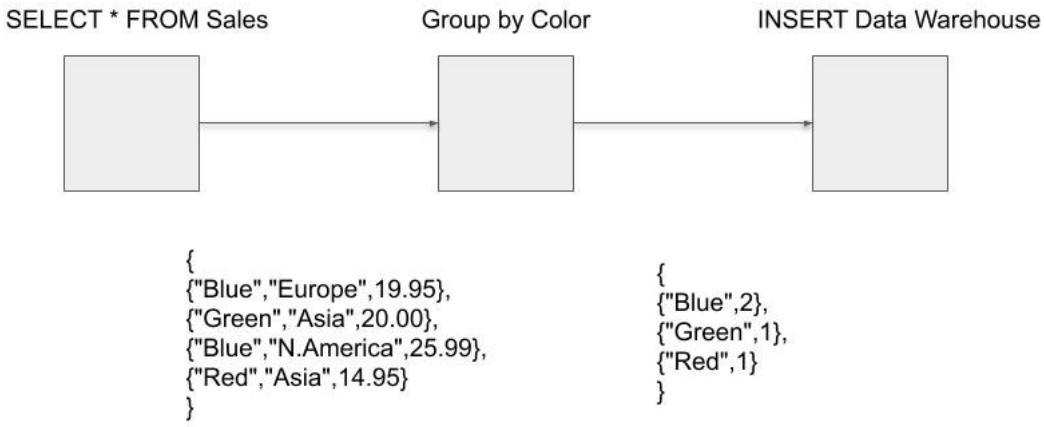

Figure 1.4 – A DAG showing the flow of data between nodes. The task follows the arrows (is directed) from left to right

This book will cover the basics of Apache Airflow but will primarily use Apache NiFi to demonstrate the principles of data engineering. The following is a screenshot of a DAG in Airflow: